Tunnels sécurisés pour environnements contraints

Loïc Ferreira

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

INSA RENNES

THESE DE DOCTORAT DE

L'INSA RENNES
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

Loïc FERREIRA

Tunnels sécurisés pour environnements contraints

Thèse présentée et soutenue à Rennes, le 18 novembre 2019
Unité de recherche : IRISA
Thèse N° : 19ISAR 18 / D19 - 18

Rapporteurs avant soutenance :

Céline Chevalier	Maître de conférences, Université Panthéon-Assas Paris 2
Henri Gilbert	Directeur du laboratoire de cryptographie, ANSSI

Composition du Jury :

Président	
Serge Vaudenay	Professeur, EPFL
Examinateurs	
Céline Chevalier	Maître de conférences, Université Panthéon-Assas Paris 2
Henri Gilbert	Directeur du laboratoire de cryptographie, ANSSI
Gildas Avoine	Professeur, INSA Rennes
Sébastien Canard	Ingénieur de recherche, Orange Labs
Caroline Fontaine	Directrice de recherche CNRS, ENS Paris-Saclay
María Naya-Plasencia	Directrice de recherche Inria, Inria Paris
David Pointcheval	Directeur de recherche CNRS, ENS
Directeur de thèse	
Gildas Avoine	Professeur, INSA Rennes
Co-directeur de thèse	
Sébastien Canard	Ingénieur de recherche, Orange Labs

Intitulé de la thèse :

Secure Tunnels for Constrained Environments

Loïc FERREIRA

En partenariat avec :

Secure Tunnels for Constrained Environments

Loïc Ferreira

Supervisors: Gildas Avoine and Sébastien Canard

In memoriam
Maria da Luz

Résumé en français

Établir une communication sécurisée entre des entités distantes est l'un des objectifs auxquels la cryptographie cherche à répondre. De tels tunnels sécurisés impliquent la mise en œuvre d'algorithmes cryptographiques de nature (symétrique, asymétrique) et de complexité différentes. Le besoin de communications sécurisées se manifeste particulièrement avec l'apparition d'objets connectés aux usages très divers et la multiplication des interactions entre ces objets.

Le développement de l'Internet des Objets entraîne le déploiement accéléré d'objets dits « à bas coût ». Contrairement à un ordinateur personnel ou un smartphone, ces objets ont des capacités très limitées notamment en termes de calcul, de communication et d'alimentation en énergie. Néanmoins, ces objets participent à la gestion d'infrastructures et d'équipements parfois très sensibles (fourniture d'eau, d'électricité, pacemaker ou défibrillateur implanté dans le corps humain, etc.). Les données échangées (dont les commandes reçues) par ces objets doivent donc être protégées à la hauteur de ces usages. Cela requiert un haut niveau de sécurité, permis en général par des mécanismes cryptographiques de relativement grande complexité calculatoire. Or les algorithmes usuellement implémentés sur un ordinateur ou un smartphone ne sont pas fonctionnels sur des objets connectés étant données les capacités réduites de ces derniers.

Cette partie introduit les problématiques des protocoles de sécurité destinés aux objets à bas coût. Elle présente également les résultats obtenus au cours de ce travail de doctorat, dont l'objectif est d'analyser la sécurité de protocoles existants et de produire des mécanismes d'échange de clé applicables aux objets à bas coût, sans compromis entre sécurité et efficacité.

Contexte

L'un des buts de la cryptographie est de permettre à deux entités distantes de communiquer de manière sécurisée. Cet objectif est rempli quotidiennement et de manière transparente lorsque l'on utilise un ordinateur personnel ou un smartphone pour, par exemple, consulter un service bancaire ou parler avec une personne éloignée. La mise en œuvre d'un tel tunnel sécurisé est rendue possible par l'utilisation conjointe d'algorithmes cryptographiques qui ont chacun une fonction différente. La cryptographie asymétrique intervient alors lors d'une phase qui précède la communication proprement dite. Les algorithmes asymétriques sont utilisés pour que les deux parties impliquées puissent mutuellement s'authentifier (*i.e.* : avoir la garantie de leur identité respective) et permettre de partager des paramètres secrets. Ces paramètres sont ensuite manipulés par des algorithmes symétriques qui vont concrètement protéger les messages (par exemple vocaux, visuels) échangés entre les deux parties. Ils garantissent que ces messages ne sont accessibles qu'aux deux parties légitimes (propriété de confidentialité) et qu'il est impossible de modifier ces messages à l'insu des deux parties prenant part à la

communication (propriété d'intégrité).

Ces algorithmes qui remplissent des fonctions distinctes ont des caractéristiques techniques différentes. D'une manière générale, les algorithmes symétriques sont beaucoup plus rapides que les algorithmes asymétriques. Mais, comme indiqué plus haut, les algorithmes asymétriques permettent de garantir des propriétés de sécurité que ne peuvent offrir les algorithmes symétriques. C'est la raison pour laquelle les deux types d'algorithmes sont généralement utilisés dans les protocoles de sécurité : bénéficiant pleinement de leur complémentarité, ces protocoles atteignent un niveau de sécurité supérieur. L'une des propriétés de sécurité, fondamentale, rendue possible par la cryptographie asymétrique est la *confidentialité persistante* (ou *forward secrecy* [Gün90; DvW92]) généralement obtenue par le protocole Diffie-Hellman (DH) [DH76]. Cette propriété garantit que la divulgation d'un paramètre secret permanent ne compromet pas la sécurité des communications effectuées antérieurement à cette divulgation. Cela permet donc de maintenir la sécurité des communications passées en dépit de la compromission d'un paramètre secret important et réduit l'étendue des conséquences d'une telle compromission.

Avec l'émergence de l'Internet des Objets (*Internet of Things* ou IoT), une multitude d'« objets connectés » sont déployés. Ces objets à bas coût de production ont des capacités de calcul et de communication restreintes. De même ils disposent d'une ressource limitée en termes d'alimentation électrique (ainsi ils peuvent être alimentés à l'aide d'une batterie dont il s'agit d'économiser la consommation, voire ne recevoir d'énergie que lorsqu'ils entrent en communication avec un lecteur). Parmi les cas d'usage impliquant ces objets, on peut citer les réseaux de senseurs sans fil (*Wireless Sensor Networks* ou WSN), la radio-identification (*Radio Frequency Identification* ou RFID), les cartes à puce, les unités de contrôle véhiculaires (*Controller Area Network* ou CAN), la domotique, l'IoT industriel et la téléphonie mobile.

Ces objets participent à la gestion ou au fonctionnement d'infrastructures ou d'équipements qui rendent des services sensibles tels que la fourniture d'eau, d'électricité ou l'assistance médicale par le biais de pacemakers et de défibrillateurs implantés dans le corps humain. Les commandes transmises à ces objets et les données récupérées auprès de ces derniers impliquent donc un niveau de sécurité à la mesure du service qu'ils contribuent à rendre. Or le niveau de sécurité d'un algorithme cryptographique est généralement lié à sa complexité. Si les ordinateurs personnels et les smartphones peuvent exécuter des algorithmes cryptographiques « lourds » et complexes, il n'en est pas de même des objets connectés. Se pose alors, pour ces objets, la difficulté de résoudre la contradiction induite par une attente élevée en termes de sécurité et une faible capacité en termes de fonctionnalité et d'efficacité.

Un champ de la cryptographie s'attache, approximativement depuis le début du millénaire, à concevoir des algorithmes cryptographiques fonctionnels pour des objets à bas coût. La grande majorité de ces algorithmes sont symétriques. S'agissant de la cryptographie asymétrique, à part quelques exceptions, les efforts tendent, avec des succès mitigés, à optimiser l'implémentation d'algorithmes usuels afin de les rendre fonctionnels sur ces objets. Ces travaux constituent une étape importante et nécessaire. Néanmoins, l'établissement d'un tunnel sécurisé suppose plus de fonctionnalités et de propriétés de sécurité qu'un simple algorithme de chiffrement ou de hachage.

Objectifs

Etant données les capacités réduites des objets connectés, le choix peut être fait de réduire les fonctionnalités des protocoles de sécurité existants et de choisir des mécanismes cryptographiques peu « coûteux » afin que le résultat soit implémentable et fonctionnel dans un objet disposant de faibles capacités. Cette démarche, bien qu'intéressante, n'est toutefois pas com-

plètement satisfaisante puisqu'elle pose notamment les questions de la souplesse du protocole ainsi obtenu et celle de la sécurité globale offerte.

Ainsi, les protocoles existants destinés à l'établissement de communications sécurisées avec ces objets s'appuient sur deux principes généraux. Tout d'abord une seule fonction symétrique est mise en œuvre. Ensuite la sécurité s'appuie sur un unique paramètre secret, exploité par l'objet tout au long de sa vie. Cela ne permet pas de garantir les mêmes propriétés et donc le même niveau de sécurité rendus possibles par l'utilisation additionnelle de mécanismes cryptographiques asymétriques. En particulier, la divulgation de cette clé symétrique permanente aboutit à la compromission de toutes les communications futures mais aussi passées que l'objet a établies.

L'objectif principal de cette thèse est de concevoir des protocoles d'échange de clé, destinés à permettre l'établissement de tunnels sécurisés entre objets à bas coût, offrant un niveau de sécurité plus élevé que les protocoles existants tout en étant fonctionnels sur ces objets. Nous tentons de tirer profit des caractéristiques techniques intrinsèques des algorithmes symétriques pour aboutir à ce résultat en refusant le compromis entre sécurité et efficacité.

Contributions

Dans cette section nous décrivons brièvement les résultats obtenus au cours de ce doctorat. Ces résultats peuvent se classer en les trois catégories suivantes :

- Nous avons procédé à l'analyse de protocoles symétriques (dont des protocoles déployés à grande échelle dans le monde) et montré qu'ils souffrent de vulnérabilités. Ces défauts de conception permettent la mise en œuvre d'attaques (quasiment) pratiques. Nous en présentons une démonstration concrète en attaquant avec succès, dans un cadre expérimental, des cartes à puce implémentant l'un des protocoles analysés.
- Nous avons conçu des modèles de sécurité que nous avons ensuite utilisés afin d'analyser rigoureusement les protocoles que nous avons construits.
- Nous avons élaboré des protocoles symétriques d'échange de clé authentifié impliquant deux ou trois parties. Ces protocoles offrent des garanties de sécurité supérieures aux protocoles existants. Ces propriétés et fonctionnalités additionnelles que nous avons obtenues incluent notamment la propriété fondamentale de confidentialité persistante et aussi la reprise de session. Cette dernière est particulièrement avantageuse pour des terminaux aux ressources limitées en termes de communication et calcul.

Les articles acceptés en conférence (et, pour la plupart d'entre eux, déjà présentés) au cours de ce doctorat sont les suivants :

[ACF19] Gildas Avoine, Sébastien Canard, and Loïc Ferreira. *IoT-Friendly AKE: Forward Secrecy and Session Resumption Meet Symmetric-Key Cryptography.* In: *ESORICS 2019, Part II.* Ed. by Kazue Sako, Steve Schneider, and Peter Y. A. Ryan. Vol. 11736. LNCS. Springer, Heidelberg, Sept. 2019, pp. 463–483.

[ACF20] G. Avoine, S. Canard, and L. Ferreira. *Symmetric-key Authenticated Key Exchange (SAKE) with Perfect Forward Secrecy.* In: *CT-RSA.* (To appear). 2020.

[AF18a] Gildas Avoine and Loïc Ferreira. "Attacking GlobalPlatform SCP02-compliant Smart Cards Using a Padding Oracle Attack". In: *IACR TCHES* 2018.2 (2018). https://tches.iacr.org/index.php/TCHES/article/view/878, pp. 149–170. ISSN: 2569-2925.

[AF18b] Gildas Avoine and Loïc Ferreira. *Rescuing LoRaWAN 1.0*. In: *FC 2018*. Ed. by Sarah Meiklejohn and Kazue Sako. Vol. 10957. LNCS. Springer, Heidelberg, 2018, pp. 253–271.

[CF19] Sébastien Canard and Loïc Ferreira. *Extended 3-Party ACCE and Application to LoRaWAN 1.1*. In: *AFRICACRYPT 19*. Ed. by Johannes Buchmann, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 11627. LNCS. Springer, Heidelberg, July 2019, pp. 21–38.

Analyse cryptographique de protocoles symétriques [AF18b; AF18a; CF19]

Analyse du protocole LoRaWAN. Le premier protocole de sécurité considéré est LoRaWAN. La version 1.0 de ce protocole est actuellement une norme de fait pour les réseaux IoT longue distance et à basse consommation (*Low Power Wide Area Network* ou LPWAN) et est déployé dans plus de cent pays dans le monde entier. L'analyse approfondie que nous avons effectuée montre que ce protocole souffre de vulnérabilités. Nous décrivons précisément comment ces défauts peuvent être exploités pour mettre en œuvre différents types d'attaque, y compris des attaques pratiques. Ces attaques permettent d'enfreindre l'intégrité et la confidentialité des données applicatives et de rompre la disponibilité d'un réseau.

Le premier type d'attaque aboutit à « désynchroniser » un terminal vis-à-vis du réseau (*i.e.* : le terminal est déconnecté du réseau). Le deuxième type d'attaque permet de rejouer et de déchiffrer des trames applicatives (sans connaissance de la clé correspondante). Cela permet donc de tromper des éléments essentiels du réseau (situés dans le cœur de réseau) ou le terminal distant. Ces attaques, dues aux défauts intrinsèques du protocole, ne s'appuient pas sur des défauts d'implémentation ou matériels. Elles sont applicables, avec forte probabilité, contre tout équipement implémentant le protocole LoRaWAN 1.0.

Finalement, nous présentons des recommandations permettant de contrecarrer les attaques décrites tout en étant compatibles avec la spécification et en maintenant l'interopérabilité entre un équipement corrigé et un équipement original. Selon nous, ces contre-mesures peuvent être aisément mises en œuvre.

Le protocole LoRaWAN 1.1 se destine à être le successeur de la version 1.0. Il a pour objectif de pallier les faiblesses de la version 1.0. Nous présentons des vulnérabilités encore présentes dans cette nouvelle version 1.1. On peut notamment citer la possibilité d'épuiser des paramètres de taille trop petite et donc d'interdire à un terminal de se connecter au réseau distant, de contraindre un terminal en version 1.1 à exécuter la version 1.0, ce qui permet d'exploiter les faiblesses de cette dernière, ou encore de rejouer et de déchiffrer des messages chiffrés.

Ces vulnérabilités sont dues essentiellement au fait de s'appuyer conceptuellement sur la précédente version et de vouloir garantir une certaine compatibilité entre les deux versions. Elles sont également induites par l'introduction d'une entité supplémentaire dans l'architecture LoRaWAN : un serveur utilisé comme tiers de confiance et jouant principalement le rôle de gestionnaire de clés. Là encore nous présentons des recommendations permettant de pallier ces différentes vulnérabilités.

Analyse du protocole SCP02. Le second protocole analysé est le protocole de sécurité SCP02. Ce protocole est déployé dans plusieurs milliards de cartes à puce (dont les cartes SIM pour la téléphonie mobile). Nous montrons que le protocole SCP02 est sujet à une attaque de type *padding oracle attack* qui permet de déchiffrer, sans connaissance de la clé correspondante, des données applicatives transmises dans un tunnel sécurisé par SCP02. Nous fournissons les résultats de nos expérimentations pratiques obtenues à partir de 10 modèles différents de cartes

à puce conçus par six encarteurs. A notre connaissance, il s'agit de la première attaque réalisée avec succès contre le protocole SCP02.

Nous récapitulons également un ensemble de méthodes permettant de contrecarrer cette attaque et en proposons de nouvelles applicables dans le cas de SCP02.

Cette attaque est possible pour deux raisons. Elle est tout d'abord rendue possible par les caractéristiques techniques intrinsèques du protocole (*i.e.* : l'ordre des opérations cryptographiques lors du déchiffrement d'un message). Ensuite l'implémentation du protocole dans les cartes à puce que nous avons testées fournit le canal auxiliaire permettant la mise en œuvre concrète de l'attaque. A ce stade, nous ne pouvons préciser si ce comportement différencié de la carte s'explique par le code source du protocole ou par les contraintes techniques du composant qui l'implémente.

Ces résultats, exploitant une technique d'attaque décrite en 2002, montre que la sécurité d'un mécanisme cryptographique (et de tout produit qui l'implémente) a une date d'expiration et doit être périodiquement analysé de nouveau à la lumière de l'état de l'art en cryptographie et en sécurité.

Ces différents résultats ont été présentés à chaque consortium en charge de la spécification technique du protocole analysé. LoRa Alliance a produit un document de recommandations destiné à renforcer la sécurité des réseaux LoRaWAN 1.0. Par ailleurs, certains changement ont également été pris en compte dans la spécification de LoRaWAN version 1.1. Concernant SCP02, ces résultats ont contribué à la décision de GlobalPlatform de déconseiller officiellement l'usage du protocole, considéré comme obsolète. Par ailleurs, les fabricants des modèles de carte analysés ont eu communication de nos travaux préalablement à leur présentation en conférence (entre octobre 2017 et mars 2018).

Ces travaux ont été présentés lors des conférences internationales *Financial Cryptography and Data Security* (FC, 2018), *Cryptographic Hardware and Embedded Systems* (CHES, 2018) et *Africacrypt* (2019). Ils sont décrits de manière détaillée dans les chapitres 3 et 4.

Conception de modèles de sécurité [CF19; ACF19]

Modèle de sécurité 3-ACCE. Nous avons étendu la notion de modèle de sécurité 3-ACCE et produit un cadre d'analyse qui permet de saisir les propriétés de sécurité que les protocoles à trois parties doivent, selon nous, garantir afin d'établir des tunnels sécurisés entre plusieurs entités.

Nous avons appliqué ce modèle au protocole LoRaWAN 1.1. Par un choix de paramètres et de déploiement appropriés nous décrivons une version légèrement modifiée de LoRaWAN 1.1 et prouvons formellement que cette version est sûre dans notre modèle 3-ACCE.

Modèle de sécurité 3-AKE. Nous avons également conçu un modèle de sécurité permettant d'analyser les protocoles d'échange de clé authentifié mis en œuvre entre trois parties. Ce modèle inclut la propriété de sécurité fondamentale de confidentialité persistante.

Ces différents travaux s'inscrivent dans le champ des protocoles tripartites et de leurs modèles de sécurité. Comme l'attaque théorique et les vulnérabilités contre le protocole LoRaWAN 1.1 l'illustrent (décrites dans le chapitre 3), ces protocoles supposent une analyse de sécurité rigoureuse dans un cadre permettant de saisir des menaces fortes.

Ces travaux s'inscrivent dans une démarche de meilleure compréhension et d'analyse de la sécurité de protocoles multipartites, lesquels reflètent la complexité croissante des communica-

tions et des interactions générées par l'IoT.

Ces travaux ont été présentés lors des conférences internationales *Africacrypt* (2019) et *European Symposium on Research in Computer Security* (ESORICS, 2019). Ils sont décrits plus précisément dans le chapitre 5.

Conception de protocoles symétriques d'échange de clé authentifié [ACF20; ACF19]

Protocole SAKE. Nous décrivons un protocole d'échange de clé authentifié entre deux parties, que nous appelons SAKE. Bien que basé uniquement sur des fonctions symétriques, ce protocole garantit la propriété de confidentialité persistante sans besoin de procédure additionnelle (telle qu'une phase de resynchronisation) ou de fonctionnalité supplémentaire (telle qu'une horloge synchronisée). L'idée sous-jacente est d'actualiser la clé symétrique principale à l'issue de chaque session. Nous résolvons le problème de la synchronisation entre les deux parties qui se pose alors avec un mécanisme simple et efficace.

Le protocole SAKE garantit que, quel que soit l'état des deux parties en termes de synchronisation lors du démarrage d'une session, ces deux parties partagent une nouvelle clé de session et sont synchronisées une fois la session conclue correctement : notre protocole SAKE est auto-synchronisant. De même qu'avec un protocole basé sur des fonctions asymétriques (par exemple DH), SAKE permet d'effectuer un nombre virtuellement illimité de sessions. De plus nous prouvons formellement la sécurité du protocole.

Nous décrivons également une variante du protocole, que nous appelons SAKE-AM. Utilisée conjointement avec SAKE, cette variante permet d'obtenir une implémentation qui hérite des différentes propriétés de SAKE (notamment la propriété de confidentialité persistante). Cette implémentation permet à n'importe laquelle des deux parties d'initier une session de telle sorte que la moindre part de calculs soit toujours réalisée par la même partie. Cette implémentation est particulièrement avantageuse dans le contexte de l'IoT où des terminaux à bas coût communiquent avec un serveur central disposant de ressources en termes de calcul notoirement plus importantes.

A notre connaissance, SAKE est le premier protocole garantissant la propriété de confidentialité persistante et basé sur des fonctions cryptographiques symétriques qui soit comparable au protocole DH (au-delà des différences intrinsèques entre cryptographie symétrique et cryptographie asymétrique).

Protocole 3-AKE. Nous présentons également un protocole générique d'échange de clé authentifié à trois parties pour l'IoT que nous appelons 3-AKE. Ce protocole est basé exclusivement sur des fonctions symétriques (pour ce qui concerne les échanges entre le terminal distant et le serveur central) et garantit la propriété de confidentialité persistante, à la différence de protocoles IoT similaires (dont ceux déployés à grande échelle). Ce protocole permet d'effectuer des reprises de session sans réduire la sécurité (en particulier la confidentialité persistante est toujours garantie). Cette fonctionnalité permet de réduire les coûts de calcul et de communication et est donc particulièrement avantageuse pour les terminaux à faibles ressources.

Notre protocole 3-AKE peut être mis en œuvre dans un contexte de déploiement IoT réel (impliquant une multitude de terminaux distants et de serveurs) de telle sorte que ce dernier bénéficie des propriétés intrinsèques du protocole. Cela permet notamment à des terminaux (mobiles) de se connecter de manière sécurisée à des serveurs successifs à un coût (en termes de calcul et de communication) réduit et sans compromettre la sécurité des échanges effectués avec de précédents serveurs.

Ces travaux ont été présentés lors de la conférence internationale *European Symposium on Research in Computer Security* (ESORICS, 2019). Ils sont décrits plus précisément dans les chapitres 6 et 7.

Conclusion

Au cours de ce doctorat nous avons étudié les moyens permettant l'établissement d'un tunnel sécurisé par des terminaux disposant de peu de ressources en termes de calcul, de communication et d'énergie notamment.

Au départ de notre démarche, nous avons constaté que la plupart des protocoles de sécurité dédiés à l'IoT, basés sur des fonctions cryptographiques symétriques, ne garantissent pas des propriétés de sécurité fortes telles que la confidentialité persistante. D'autres protocoles, s'appuyant sur des mécanismes asymétriques, ne sont pas fonctionnels sur ces terminaux. D'une manière générale, la plupart de ces protocoles privilégient l'efficacité au détriment de la sécurité (chapitre 1). Nous avons illustré (concrètement) ce constat par l'analyse de deux protocoles de sécurité largement déployés (chapitres 3 et 4). Dans un deuxième temps, nous avons recueilli et produit (lorsqu'ils faisaient défaut) les outils méthodologiques qui nous sont apparus nécessaires à la saine conception des mécanismes qui sont l'un des objectifs finaux de notre travail (chapitre 5). Finalement, nous avons proposé de nouveaux protocoles d'échange de clé, présentant des propriétés accrues (en termes de sécurité et d'efficacité) relativement aux protocoles existants (chapitres 6 et 7). A présent nous considérons les perspectives ouvertes par les travaux entrepris au cours de ce doctorat et les voies qui demandent à être poursuivies.

Perspectives et questions ouvertes

Approfondissements. Le protocole SAKE (resp. SAKE-AM) décrit dans le chapitre 6 est constitué de cinq (resp. quatre) messages qui peuvent être réduits à quatre (resp. trois) si les deux parties prenantes sont synchronisées (relativement à leurs clés maîtres) au démarrage de la session. Chacun des messages du protocole dessert un objectif particulier : authentifier les deux parties, détecter un décalage, rétablir la synchronisation. L'ensemble permet finalement d'atteindre la propriété de confidentialité persistante. La suppression d'un message ouvre la possibilité d'une attaque, comme l'ont montré les nombreuses versions alternatives que nous avons explorées. Nous pensons que ce nombre de cinq messages est le moindre possible mais n'avons pas formellement répondu à cette question de l'optimalité.

Au chapitre 5, nous présentons un modèle de sécurité 3-AKE. Il est utilisé pour analyser le protocole d'échange de clé à trois parties décrit au chapitre 7. Ce modèle définit la sécurité notamment sur la notion d'indistinguabilité des clés de session. Le calcul de la clé de session « finale » suppose l'implication, dans des calculs préalables, de clés de session intermédiaires. Cela entre en contradiction avec le paradigme d'indistinguabilité des clés. Pour pallier cette difficulté, nous avons limité les possibilités pour l'adversaire de tester extensivement les différentes clés. Un modèle de sécurité plus approprié permettrait à la fois de conserver cette notion d'indistinguabilité et de ne pas restreindre les actions de l'adversaire. A cet égard, les modèles de Brzuska, Fischlin, Warinschi et Williams [BFWW11], Brzuska, Fischlin et Smart [BFS+13] ou Krawczyk [Kra16] peuvent servir d'inspiration.

Finalement, procéder à l'implémentation des protocoles SAKE et 3-AKE sur des terminaux à bas coût permettrait de montrer concrètement leur efficacité.

Confidentialité persistante en cryptographie symétrique. Un équivalent symétrique du protocole Diffie-Hellman remplacerait avantageusement ce dernier. Quelques travaux ont été effectués en ce sens par le biais de généralisations algébriques du schéma DH [Par15; PN18] ou avec la notion de « fonction de conversion » [CK05] (construite à partir d'une fonction symétrique, une telle fonction permet de transformer le chiffré correspondant à une certaine clé en le chiffré correspondant à une autre clé). Ce champ a, pour l'instant, aboutit à des résultats mitigés et mérite d'être défriché.

Cryptographie post-quantique. L'un de nos objectifs a été de concevoir des protocoles d'échange de clé à la fois efficaces sur un terminal à bas coût et présentant de meilleurs propriétés de sécurité que les protocoles existants. Nous avons donc proposé des mécanismes basés sur des fonctions symétriques. Avec l'ère de la cryptographie post-quantique il est maintenant clair que les schémas cryptographiques asymétriques classiques les plus courants sont cassés. Si la plupart des fonctions symétriques semblent encore préservées, des menaces émergent contre certaines primitives [KLLN16a; CNS17; KLLN16b].

Ainsi, certaines fonctions symétriques (chiffrement, MAC) sûres dans un modèle classique ne sont pas résistantes dans un modèle quantique. En effet, à partir de ces primitives il est possible de définir des fonctions faisant apparaître une structure interne (par exemple une période cachée) qui peut être révélée par la troublante magie de la mécanique quantique. Cela permet alors d'accéder à un paramètre secret (tel qu'une clé) de l'algorithme attaqué.

L'étude de la sécurité quantique peut être prolongée des primitives aux protocoles. Au-delà de la possibilité de s'attaquer aux primitives qui constituent un protocole, on peut essayer de déterminer dans quelle mesure les caractéristiques techniques internes de ce dernier peuvent être exploitées pour l'attaquer. Plus généralement, la cryptanalyse quantique des primitives et des protocoles cryptographiques (y compris ceux que nous proposons dans cette thèse) est à poursuivre et à entreprendre, de même que la conception de modèles de sécurité appropriés au contexte quantique.

« Survivabilité » d'un protocole de sécurité. Ce qui peut surprendre de la part de certains protocoles IoT existants est la possibilité de les casser, parfois trivialement. Cela est dû aux caractéristiques intrinsèques à ces protocoles. Ainsi, les attaques contre le protocole LoRaWAN 1.0 décrites au chapitre 3 reposent essentiellement sur la taille extrêmement réduite de certains de ses paramètres. Les concepteurs de LoRaWAN justifient ce choix par le nombre limité d'échanges de clé qu'un terminal est censé effectuer au cours de sa vie. Néanmoins, nous avons montré comment contraindre un terminal à initier beaucoup plus de sessions que prévu, ce qui aboutit à des conséquences néfastes en termes de sécurité. Cet exemple conduit à nous interroger sur le comportement d'un protocole lorsque ce dernier n'est pas mis en œuvre dans les conditions requises par ses concepteurs.

La notion que nous abordons ici ne correspond pas à la *résilience* ou à la *performabilité*. La première notion peut être définie comme la capacité d'un système attaqué ou en présence de fautes à repasser d'un état dégradé à un état nominal [CMH+07]. La seconde permet d'apprécier comment un système peut résister et maintenir ses fonctionnalités nominales en présence d'attaques [Mey80; Mey92]. La notion que nous tentons de saisir se rapproche de celle de *survivabilité* [KSS03; SKH+02]. Intuitivement, il est souhaitable que les propriétés de sécurité « essentielles » d'un système soient préservées, ce qui implique de définir à quoi celles-ci correspondent.

Le développement de l'IoT entraîne le déploiement d'un grand nombre de terminaux physiquement accessibles à l'attaquant et pour lesquels il n'y a pas de moyen simple permettant une

mise à jour sécurisée. Le concept de survivabilité peut donc utilement contribuer à la sécurité de ces terminaux et de leurs utilisateurs.

Contents

1 Introduction

Secure tunnel establishment is one of the main task of authenticated key exchange protocols. These protocols aim at sharing a secret key material among two or more parties in order to subsequently exchange data in a secure manner.

With the arrival of the Internet of Things, several security protocols have been proposed which aim at being implemented in devices with different capabilities in terms of computation, communication, and energy. This list includes (very) constrained devices. Therefore, these protocols must face the following challenge: being fully functional, and at the same time providing the highest security level as possible.

In this chapter, we present the motivations and difficulties behind authenticated key exchange protocols dedicated to low-resource devices, and the methods used to analyse and formally prove the security of such protocols. We also describe different solutions which have been proposed in the past. Finally we present the contributions of this thesis.

Contents

1.1 Motivation

1.1.1 The Rise of Computing Machines and Cryptography

Symmetric and asymmetric cryptography. The first and still a paramount goal of cryptography is to ensure confidentiality of data. All along the history of mankind, different techniques have been devised to achieve this task. Some of them aimed at hiding the existence of the data, others aimed at hiding the meaning of the data. Since the birth of modern cryptography after World War II, new schemes have been conceived in order to ensure the confidentiality of data. Applying the Kerckhoffs principle [Ker83a; Ker83b], these schemes are based on (simple) mathematics operations and make use of one unique secret parameter: the encryption key. The same secret key is used to encrypt the data and decrypt the ciphertext. This paradigm is called *symmetric cryptography*. To the best of our knowledge, the first modern examples of such schemes are Lucifer [Fei73] and DES [Nat99].

Data confidentiality is a crucial need when a communication occurs over an insecure channel. In such a context, prior to exchanging encrypted data, the secret key must be shared between the two parties involved in the communication. The issue of transmitting a secret key over an insecure channel has been given a solution with the seminal paper of Diffie and Hellman (DH) [DH76] which founded the field of *asymmetric cryptography*. The basic idea consists in using two keys instead of one: a public key which can be surrendered to anyone, and a private key which must be known only to the legitimate party. Hence, any distant party B willing to communicate with some party A can use A's public key to send or compute a secret value that A can yield in turn with its private key. Shortly after, Rivest, Shamir and Adleman (RSA) [RSA78] presented an asymmetric encryption (and a signature) scheme. Nowadays, with the sole purpose of key agreement (putting aside the question of authenticating the parties, necessary to ensure that the secret key is shared only by legitimate parties), these asymmetric schemes are extensively used in widely deployed protocols such as TLS 1.2 [DR08], TLS 1.3 [Res18], IKEv2 [KHN+14], SSH [YL06c; YL06a; YL06d; YL06b] to name a few.

Smaller, more pervasive computing devices. The need for and the development of modern cryptographic schemes has evolved with the field of computer science. The Colossus and ENIAC machines, at the time of World War II, can be regarded as the first electronic, digital computers. They opened the first era of computing: the mainframe computers owned and used by governmental agencies or companies. Followed the era of mini and then personal computers (with machines such as the Hewlett-Packard HP 3000, the R2E Micral, or the DEC PDP-8), where the latter are owned and used by one person. With the invention of the first micro-processor by Intel in 1971, the computers become smaller and faster. This opens the era of ubiquitous computing characterised by the widespread use of small computer products, such as personal digital assistants or smartphones [Kru09]. Following the prediction of Mark Weiser [Wei91] that "*the most profound technologies* [...] *weave themselves into the fabric of everyday life until they are indistinguishable from it*", the rise of the Internet of Things (IoT) leads to the deployment of more pervasive and also more constrained devices with respect to their capabilities.

It is expected that 125 billion connected objects be deployed by 2030 [IHS17]. They are nowadays used in various contexts, ranging from Wireless Sensor Networks (WSNs), Radio Frequency Identification (RFID) tags, smart cards, Controller Area Networks (CANs) for vehicular systems, smart home, medical care (eHealth), up to industrial Internet of Things. Compared to standard computing devices (e.g., personal computers, smartphones) these objects do not have the same level regarding energy resource, computation power, and memory size. Consequently, they cannot all efficiently implement asymmetric schemes which imply an heavier circuitry (e.g.,

modular exponentiations, operations on elliptic curve involving big numbers) than symmetric functions (essentially based on simple arithmetic operations, and small look-up tables).

Constrained devices and security. Very concretely, newspapers or research results in the academic field regularly show the security level that these constrained devices do (not) attain. To mention only a few, Miller and Valasek [MV15] were able to mount a remote attack against a Jeep Cherokee, which affects certain physical systems such as steering and braking. They guessed a WiFi password used to protect the firmware update mechanism of the multimedia system, and exploited the fact that this system is connected to the vehicle's Controller Area Network which connects multiple electronic units used in the car's functioning and driving. Ronen, Shamir, Weingarten and O'Flynn [RSWO17] succeeded in taking over a ZigBee network, and then to propagate a worm through the over-the-air update mechanism. They first retrieved (through a side-channel attack) the static master symmetric key shared by all nodes of the ZigBee network. Tierney [Tie18] has also shown how to compel a Z-Wave device to downgrade from version S2 to S0, which allows exploiting a major weakness of that version. In version S0, the network's master symmetric key can be computed by an attacker merely by eavesdropping on data exchanged when a node joins the Z-Wave network. Tierney has illustrated his attack on the keyless Yale Conexis L1 smart lock. Halperin, Heydt-Benjamin, Ransford, Clark, Defend, Morgan, Fu, Kohno and Maisel [HHR+08] analysed an implantable medical device that incorporates pacemaking and defibrillation functions. Besides uncovering privacy issues (the authors got access to unencrypted data related to the patient, such as name and diagnosis, stored at the implanted device), they succeeded in exhausting the battery-powered device, and to replay a command which triggers high voltage shocks (up to 138 V). These vulnerabilities pose a potential denial-of-service risk and clearly endanger the patient's life. Radcliffe [Rad11] has reversed the encoding scheme used to "encrypt" the messages exchanged between an insulin pump that delivers the proper level of insulin and the implanted wireless peripheral that estimates the blood glucose. Changing the settings corresponding to a specific patient allows delivering an incorrect amount of insulin, and plunging the patient into an hypoglycaemic state. Marin, Singelée, Garcia, Chothia, Willems and Preneel [MSG+16] reverse-engineered the proprietary communication protocol of an implantable defibrillator, and succeeded in compromising the device's availability by exhausting its battery. They also successfully replayed previous messages, which purpose is to deliver electric shocks. Marin, Singelée, Yang, Verbauwhede and Preneel [MSY+16] also reverse-engineered the proprietary communication protocol of an insulin pump, and, in addition to retrieving private information related to the patient, were able to send unauthorised commands to the insulin pump.

Highly likely, asymmetric schemes could enhance the security level of these devices, but, as said, not all of the latter are able to implement such an heavy circuitry, due to their limited resources.

A glimpse of implementation results. As an illustration let us consider some microcontrollers used by several IoT devices. We stress that these devices are not our target in this thesis. The resources they benefit correspond to an upper bound in regards to the very constrained devices. Nonetheless we use these more gifted devices, which barely succeeds in implementing efficiently asymmetric schemes, to illustrate the difficulty to achieve the same goal on even more constrained devices (for which we are unable to provide figures, for obvious reasons).

In order to highlight the difference between symmetric and asymmetric cryptography, we consider the implementation results of different operations in these two fields. Table 1.1 corresponds to a selective summary. A strict comparison is not possible due to the intrinsic differences

between symmetric and asymmetric cryptography. Rather we aim at highlighting the gap, with regards to the implementation efficiency on constrained devices, that separates these two types of operations, which are commonly executed in key exchange protocols.

We start with the Atmel ATmega128L [Atm11] (running at 7.37 MHz frequency) which equips the MICAz and MICA2 motes. A point multiplication on a 256-bit prime field elliptic curve takes 1.28 seconds (second component in Table 1.1). This leads to an ECDH key exchange which lasts 4.14 seconds [LWG14]. This figure does not include any authentication step (e.g., a signature). On the Atmel ATmega2560 chip [Atm14] (at 16 MHz frequency), an ECDH key exchange corresponding to a 255-bit prime field elliptic curve lasts 3.49 seconds [HS13]. The Ed25519 signature [BDL+12] computation and verification adds respectively 2.14 and 2.51 seconds [HS13]. In order to get a full key exchange the session key derivation step must be added. Nonetheless it involves usually only symmetric-key operations, hence it is much faster than the signed ECDH key exchange. Therefore the computation time for such a full key exchange on ATmega2560 lasts roughly 8.14 seconds (third component in Table 1.1). Furthermore, the time needed to send and receive the key exchange messages must be also added. On a MSP430 microcontroller [Tex03] (running at 8 MHz) a simple RSA 1024 decryption needs 43,368,720 cycles, and lasts 5.42 seconds [GAB19] (first component in Table 1.1).

Table 1.1 – Selective comparison between symmetric (in blue) and asymmetric (in green) operations on constrained chips

Modular exponentiation	Time[a] (cycles)	(s)
RSA 1024 decryption [GAB19]	43,368,720	5.42

Point multiplication	Time[b] (cycles)	(s)
256-bit prime field elliptic curve [LWG14]	9,420,788	1.28

Key exchange	Energy (mJ)[b]	Time (s)[b]
ECDH (Curve25519)-Ed25519 [HS13]	-	8.14
ECDH-ECDSA (160-bit prime field) [MGSP08]	283.0	-
Kerberos (AES 128) [MGSP08]	14.4	-

Symmetric encryption	Time (cycles/byte)	Energy (μJ/block)[a]	Area (GE)
AES 128 [BRS+15; MPL+11; BRS+15]	132[a]	2.18	2400[c]
Speck 128/128 [BRS+15; BSS+13; BRS+15]	103[a]	1.44	1280[d]
PRESENT 80 [DCK+15; YKPH11]	1053[b]	-	1030[c]

[a] MSP430
[b] ATmega128(L)/ATmega2960
[c] 0.18 μm
[d] 0.13 μm

In comparison, the symmetric functions perform much better. On ATmega2560, encryption with the stream cipher Salsa20 [Ber08] requires 270 cycles/byte, and authentication with

Poly1305 [Ber05] needs 177 cycles/byte [HS13].[1] This corresponds respectively to 4320 and 2832 cycles to encrypt and authenticate a 16-byte block. In contrast, a point multiplication on Curve25519 [Ber06] takes 22,791,579 cycles [HS13], and a point multiplication on a 160, 192 and 224-bit prime field elliptic curve needs respectively 6,276,630, 9,964,549, and 14,856,446 cycles [LWG14]. Regarding the implementation of lightweight block ciphers on ATmega128 [Atm11], we excerpt the following corresponding to AES 128 [Nat01], PRESENT 80 [BKL+07] and Speck 64/96 [BSS+13] (64-bit block and 96-bit key) [DCK+15]. A size less than 128 bits for the key size and the block size is not recommended in general, even though some deem that shorter parameters are acceptable in the context of IoT. Here we mention these results because a symmetric function can also be used as a building block for other purposes than encrypting or authenticating data (e.g., generating a pseudo-random value). The operation corresponding to key schedule, encryption and decryption in CBC mode of 128 bytes with AES, Speck and PRESENT needs respectively 59,085, 59,612 and 245,853 cycles. Encryption only of 128 bits in CTR mode with AES, Speck and PRESENT needs respectively 3175, 3251 and 15239 cycles. With Speck 128/128, a basic encryption operation requires 143 cycles/byte [BSS+15], which, for 16 bytes, corresponds to 2288 cycles. On a MSP430 microcontroller, encryption with AES 128 and Speck 128/128 takes respectively 132 and 103 cycles/byte (see fourth component in Table 1.1). As one can see, symmetric operations are "*lightning fast*" [TPG15] compared to asymmetric ones.

Furthermore, these devices are not only computing machines but also communicating machines. Both processes must be managed with limited energy resources. Let us consider two key exchange protocols implemented on MICAz and TelosB [MEM] (which features the MSP430 microcontroller running at 4 MHz) motes [MGSP08]. The first protocol is an ECDH-ECDSA scheme based on a 160-bit prime field elliptic curve. The second is the symmetric-key protocol Kerberos [NYHR05] with AES 128. On MICAz, ECDH-ECDSA consumes 283 mJ whereas Kerberos consumes 14.4 mJ (third component in Table 1.1). Hence, the symmetric-key protocol Kerberos is 94.9% more energy efficient than the asymmetric protocol ECDH-ECDSA. On TelosB, ECDH-ECDSA and Kerberos consumes respectively 130.9 mJ and 12.64 mJ, which corresponds to a 90.3% saving in favour of Kerberos. Moreover, the energy cost corresponding to computation and communication is distributed very differently depending on whether asymmetric or symmetric operations are executed. On MICAz, the computation process consumes 4% of the total energy cost for Kerberos and 83% for ECDH-ECDSA. A similar figure is observed on TelosB: 1% for Kerberos and 55% for ECDH-ECDSA. In contrast, on MSP430, encryption with AES 128 and Speck 128/128 consumes respectively 2.18 and 1.44 μJ per block (fourth component in Table 1.1).

More generally, efficient asymmetric operations (ECC) perform 100 to 1,000 more slowly than symmetric algorithms such as AES for instance, and this correlates with a 2 to 3 orders of magnitude higher power consumption [EKP+07].

Better, faster, lighter. Regarding asymmetric schemes, a general trend aims at improving the efficiency of their implementation, and exploiting hardware enhancements. More powerful chips and devices exist which perform better and are able to implement ECDH key exchange and other asymmetric schemes. For instance, on the MSP430 microcontroller an ECDSA signature on a 256-bit prime field elliptic curve can be computed in less than 1 second and verified in less than 2 seconds [LOL12]. However too much optimisation may yield vulnerabilities which allow for side channel attacks [NCOS16]. Furthermore, the processing units, such as microcontrollers,

[1] In the calculation of the authentication tag, Hutter and Schwabe replace the usual encryption of the nonce with the secret key itself, which is then unique per message to authenticate.

used in constrained devices have not increased their computational capabilities at the same rate as microprocessors for personal computers or smartphones [MMDF+12]. Therefore, the achievements may not be enough in order for very constrained devices to be able to implement asymmetric schemes, even optimised. These devices equip sensors and actuators which, despite (or because of) their very constrained resources (i.e., their low cost), are used in real-life deployments for home automation security, remote keyless entry, security management of infrastructures, and many other purposes that require establishing secure tunnels. In that context, there is a persistent need for smaller size, lower computation, energy, and memory resources devices, and lower production costs.

Moreover, the post-quantum era opens. Although threats emerge against some primitives [KLLN16a; CNS17; KLLN16b], symmetric cryptography seems less vulnerable against quantum computing so far. Regarding the "classic" asymmetric algorithms the assessment is straightforward: they are all broken in a post-quantum world [Sho97]. It is still uncertain if post-quantum asymmetric algorithms more efficient than the classic ones will be devised. Consequently, considering using solely symmetric cryptography and its intrinsic lightweightness in order to establish secure communications is relevant in the context of constrained devices.

1.1.2 The Need for Cryptographic Protocols Efficient on Constrained Devices

From lightweight cryptographic primitives... The development of symmetric ciphers intended to be lightweight can likely be dated from the birth of the mobile telephony with the A5/1 [BSW01] and A5/2 [BBK03] encryption stream ciphers in the late 1980s. Research on this field has become more intensive since 2010 with the proposal of numerous ciphers, mostly encryption and hash functions. Rather comprehensive surveys can be found in Biryukov and Perrin [BP17], and Hatzivasilis, Fysarakis, Papaefstathiou and Manifavas [HFPM18]. Several projects have been initiated in order to promote the design of efficient, lightweight and secure ciphers. For instance, the ECRYPT Stream Cipher Project (eSTREAM) [ECR08] organised by the European ECRYPT network between 2005 and 2008, focuses on stream ciphers, and includes a portfolio of functions intended to devices with highly restricted resources. The CAESAR competition [Ber19] dedicated to authenticated encryption (AE) functions ended in 2019 with a set of ciphers intended to resource constrained environments. In 2019, NIST started a Lightweight Cryptography competition [Nat19] which focuses on lightweight AE and hash functions.

As illustrated by the aforementioned projects, most of the lightweight cryptographic functions belong to the field of symmetric cryptography. Regarding lightweight asymmetric schemes, most of the work aims at adapting existing schemes to make them functional on a low-resource devices. Among the rare effective results, one can cite the Girault, Poupard and Stern [GPS06] (GPS) authentication and signature scheme (also known as cryptoGPS) which is an adaptation of the Schnorr scheme [Sch90], and targets as constrained devices as RFID tags. GPS has a similar limitation as the Schnorr scheme related to the usage of precomputed "coupons" that must be stored by the device. This narrows the number of GPS runs. Yet, a line of work aims at overcoming this drawback through periodic reload of coupons, or on-the-tag regeneration [HW08; NH09], as well as proposing improved variants of the protocol [GL04; CFR13].

... to lightweight cryptographic protocols. Establishing a secure tunnel between two (low-resource) parties implies implementing a protocol. The latter builds on cryptographic primitives in order to ensure several security properties (mutual authentication of the parties, confidentiality and authenticity of data). As illustrated in Section 1.1.1, a very constrained

device is unable to implement asymmetric functions, or even a wide family of ciphers each one providing distinct features (as TLS or IPsec for instance). Therefore current protocols intended to constrained devices aim at overcoming this limitation with the two main principles: firstly, one master symmetric key is shared by two (or more) parties, and secondly, a few (possibly only one) symmetric-key functions are used to ensure all functionalities and security properties.

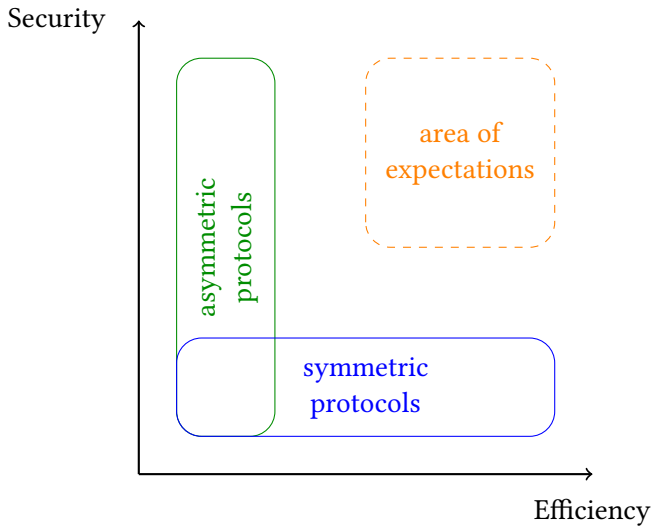


Figure 1.1 – Protocols for constrained devices: current situation and expectations

The lack of suitable symmetric-key protocols seems to mimic the evolution of block ciphers with respect to lightweight functions in the 1990s: it leads to many proprietary solutions which security is questionable (e.g., [CS15; CFPR15]). Solely built on symmetric-key functions, these protocols do not achieve the same level of security or "flexibility" provided by the protocols that make use of asymmetric schemes (see Figure 1.1). In particular, since their security is based on a static symmetric key, they lack in ensuring a fundamental property known as *forward secrecy* [Gün90; DvW92]. This property is a very strong form of long-term security which, informally, guarantees that future disclosures of some long-term secret keys do not compromise past session keys.

As illustrated in Section 1.3, several protocols imply the use of a (short) counter which limits the number of runs. Some protocols keep the ability to initiate a session for a specific party. For instance, this possibility is attributed to an end-device that connects to a central server, but the converse is not allowed. Therefore the server, which can only behave as responder, is unable to restore a problematic connection with the end-device, and must wait for the initiator to detect the issue and solve it. Meanwhile all valuable data sent by this end-device is lost. Furthermore, an additional procedure independent to the regular protocol (e.g., a resynchronisation procedure) or an extra functionality (e.g., a synchronised clock) is sometimes necessary in order to maintain the protocol's security. Such a requirement is not suitable to all constrained devices, and narrows the types of devices that can implement the protocol. Overall, the current protocols dedicated to constrained devices seemingly often *trade security for efficiency*, and *do not* attain the same security level as state-of-the-art protocols.

1.2 Basics on Key Establishment Protocols

1.2.1 Types of Key Establishment Protocols

The purpose of a *key establishment* (or key exchange) protocol is to enable two parties (possibly more) to share a secret value usually called a *session key*. In this section and in this thesis we will consider two-party protocols possibly aided with a trusted third party. In order to yield a secret key, the two parties share beforehand a common value. That can be a symmetric key or a public key certificate. In the first case, confidentiality of the symmetric key is necessary in order to guarantee the security of the key establishment protocol. Knowledge of this symmetric key allows an adversary to impersonate a legitimate party, and compute the secret value yielded by the protocol run. In the second case, integrity of the certificate is required otherwise an adversary could replace the certificate with one of her choice, which is similarly detrimental to the protocol's security.

The key establishment protocols can be broadly divided in two sets: *key transport* and *key agreement* protocols. A key transport protocol is a key establishment scheme where one party generates or obtains a session key, and then transfers it to the other party. A key agreement protocol is a key establishment mechanism where the session key is a function of input by all parties involved in the protocol run.

The two parties involved in a successful protocol run are said to be *partners*. The purpose of the session key output by a run of the key establishment protocol is usually to establish a *secure tunnel* between the two partners. This secure tunnel may then ensure *confidentiality*, *integrity* and *authenticity* of the data subsequently exchanged between the two parties. The first property guarantees that data is available only to the two partners. The second property guarantees that data has not been altered by an unauthorised entity. The third property guarantees the origin of data.

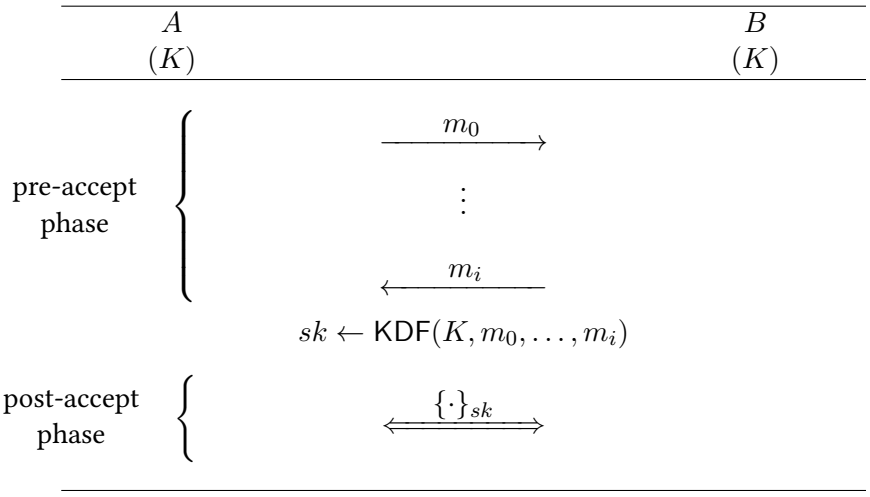


Figure 1.2 – Pre-accept and post-accept phases in a key establishment protocol. In this example, the two parties share a symmetric key K.

A key establishment protocol can be divided into two phases: a *pre-accept* phase and a *post-accept* phase. From the viewpoint of one party, the pre-accept phase ends when that party deems to be actually communicating with its intended party, and that the session key can be safely used to exchange data. This session key is used during the post-accept phase to securely exchange data between the two parties (see Figure 1.2).

1.2.2 Goals of Key Establishment Protocols

Most key establishment protocols aim at ensuring two main goals: authenticating the involved parties, and sharing a fresh secret key. This translates into two corresponding security properties: *entity authentication* and *key indistinguishability*. The first property aims at verifying that an identity is as claimed. The second property aims at ensuring that the session key output by a protocol run is indistinguishable from a value (of same size) drawn at random with the same distribution. We call *Authenticated Key Establishment* (AKE) such protocols.

What separates the pre-accept and the post-accept phases is not necessarily the moment when the session key is computed and used. In several key establishment protocols, this is the case: the session key is used only during the pre-accept phase to set up the secure tunnel. Nonetheless, there exist protocols with no clear separation between the key exchange phase and the use of the session key. This is true in particular for the Transport Layer Security (TLS) protocol in version 1.2 [DR08]. The session key is used to protect messages (called Finished) which finalise the key exchange phase (i.e., the pre-accept phase). Since these encrypted and MAC-ed Finished messages provide a trivial way to distinguish the session key from a random value, the security of TLS 1.2 cannot be proved based on indistinguishability of keys. Consequently, Jager, Kohlar, Schäge and Schwenk [JKSS11] have proposed an alternative way to capture the properties such a key establishment protocol is supposed to guarantee. The first property is entity authentication, similar to that of AKE protocols. But instead of guaranteeing that the session key is suitable to be used for any purpose, the second property ensures that the key is suitable for a specific purpose which is to set up an *authenticated and confidential channel*. More precisely, in the Jager et al.'s security model (that they call *Authenticated and Confidential Channel Establishment* or ACCE), the second property corresponds to what the secure channel itself is supposed to achieve (see Figure 1.3).

$$\text{AKE} \quad \begin{cases} \text{entity authentication} \\ \text{indistinguishability of keys} \end{cases}$$

$$\text{ACCE} \quad \begin{cases} \text{entity authentication} \\ \text{channel security} = \begin{cases} \text{indistinguishability of ciphertexts} \\ \text{authenticity of plaintexts/ciphertexts} \end{cases} \end{cases}$$

Figure 1.3 – The main properties of the AKE and ACCE security models

The variety of security models is not limited to the AKE and ACCE proposals. Multiple models have been devised which are differentiated by the rules that limit or extend what the adversary is allowed to perform (e.g., [BR94; BCK98; Sho; CK01; KOY03; Kra05; LLM07; CF12; LSY+14]). These different adversarial models allow capturing additional security properties such as resistance to key-compromise impersonation attacks [BWJM97], and also forward secrecy.

1.3 Current Protocols for Constrained Devices

In this section, we review several symmetric protocols, including widely deployed ones, and highlight their limitations. A rather comprehensive study on key exchange protocols can be found in Boyd and Mathuria [BM03a], and also in Menezes, van Oorschot and Vanstone [MVO96] (mostly in Chapter 12).

1.3.1 Overview of the Protocols

Generic protocols. The AKEP1 and AKEP2 protocols described by Bellare and Rogaway [BR94] are two efficient lightweight AKE based on two static master keys. Hence they do not provide forward secrecy. In both protocols, authentication of the parties is achieved by sending a challenge (pseudo-random values r_A, r_B) from which the responder must compute a value with one of the master keys (K'). AKEP1 is a key transport protocol where the responder B generates and sends to the initiator A a session key sk. AKEP2 (see Figure 1.4) is a key agreement protocol where the session key (sk) is computed with the other master key (K) and a pseudo-random value chosen by B.

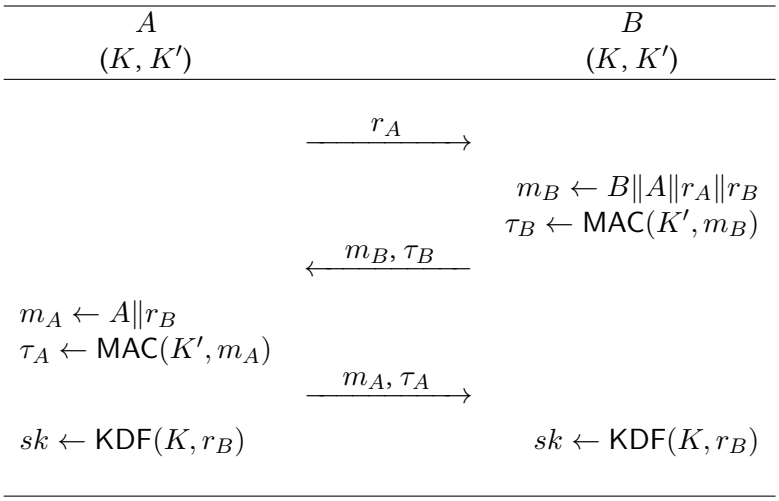


Figure 1.4 – The AKEP2 protocol

Dousti and Jalili [DJ14] describe a key agreement protocol, called FORSAKES, where the two parties use of a shared master key (K), and exchange random values (r_A, r_B) to perform the mutual authentication, and to compute the session key (sk). In order to provide forward secrecy, the master key is periodically updated (using a one-way function H). Figure 1.5 depicts the protocol.

The master key is updated regularly at each time interval. This implies a perfect *time synchronisation* between the parties. Indeed, firstly, all the operations involving both parties must be done in the same time interval. Otherwise, the parties may use different master keys, which forbids from computing a shared session key. For instance, if the time interval falls due after one party computes a MAC tag, and the other party verifies it, then either party uses a different master key. Therefore the time interval must be large enough in order for the parties be able to complete a whole protocol run. Secondly, the duration of the time interval must exactly be equal to the duration of a protocol run. Otherwise, corrupting either parties once they have accepted (but before the master key update) allows retrieving the current master key used to compute the session key, hence breaking the forward secrecy. Achieving a perfect time synchronisation may be quite complex (or even not possible) in any context, in particular with constrained devices.

We also observe that the security model chosen by Dousti and Jalili does not allow revealing the key ik (only sk) as this would allow to trivially win the security experiment based on indistinguishability of the session keys.

The protocols described in the ISO/IEC 11770-2 standard [Int08] propose different lightweight

schemes built on symmetric-key functions. Some involve two parties, other require the assistance of a third party (e.g., key server). But none provide forward secrecy.

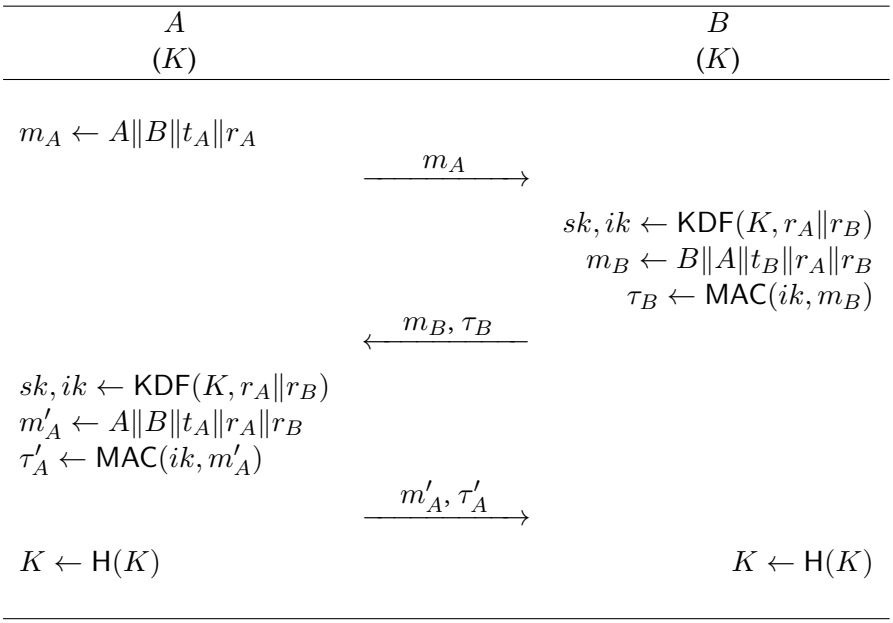


Figure 1.5 – The FORSAKES protocol. t_A and t_B correspond to timestamps.

Computer networks. Kerberos v5 [NYHR05] is a key transport protocol that enables two entities: a client A and a server B, to share a session key with the help of a trusted third party: the authentication server S (see Figure 1.6). The latter shares a distinct static symmetric key with A and B (respectively K_A and K_B). A new session key (sk) is shared as follows. Upon request from A, S sends to A a fresh session key respectively encrypted under A's and B's master key. Then A relays the encrypted session key to B, with additional data. The message received by B from A includes a validity time ℓ (encrypted by S) and a timestamp t_A (encrypted by A with the session key sk) in order to verify that the session key is not expired.

Based on static symmetric keys, Kerberos does not ensure forward secrecy. In addition, the initiator A is allowed to reuse several times the same session key (as long as it does not exceed the validity time specified by S), which increases the extent of a session key disclosure. Furthermore, the protocol implies a secure and synchronised clock (between A and B) in order to not impair its security. Since it is based on symmetric cryptography, Hardjono [Har14] proposes to use Kerberos as an IoT protocol.

The TLS 1.2 protocol [DR08] is a popular client-server protocol. The standard version is based on asymmetric schemes (to wit RSA and (EC)DH), but there exists a version built only on symmetric-key functions: the so-called pre-shared key mode (PSK) [ET05]. Figure 1.7 depicts this variant. In that case, mutual authentication and session key computation are done with the shared master key (K), and two pseudo-random values (r_C, r_S) exchanged throughout the protocol run. Urien [Uri10] has proposed to apply TLS-PSK in order to protect transactions done between a banking (EMV) card and a remote server.

Although TLS 1.3 [Res18] is not specifically dedicated to constrained devices, it also incorporates a PSK mode (see Figure 1.8). This mode essentially inherits from that of TLS 1.2 but with a shorter message flow, and different symmetric functions to authenticate the messages and derive multiple keys (in particular the handshake key hk, the finished key fk, and the session

key sk, used to protect respectively key exchange messages, the so-called Finished messages, and the application messages). In TLS 1.3 each master key has a finite lifetime but the latter can be as high as seven days. Moreover, the different master keys can be encrypted by the server with the same symmetric key (called Session Ticket Encryption Key or STEK). Therefore the disclosure of a master key or its encryption key STEK breaks forward secrecy. The same holds regarding TLS 1.2.

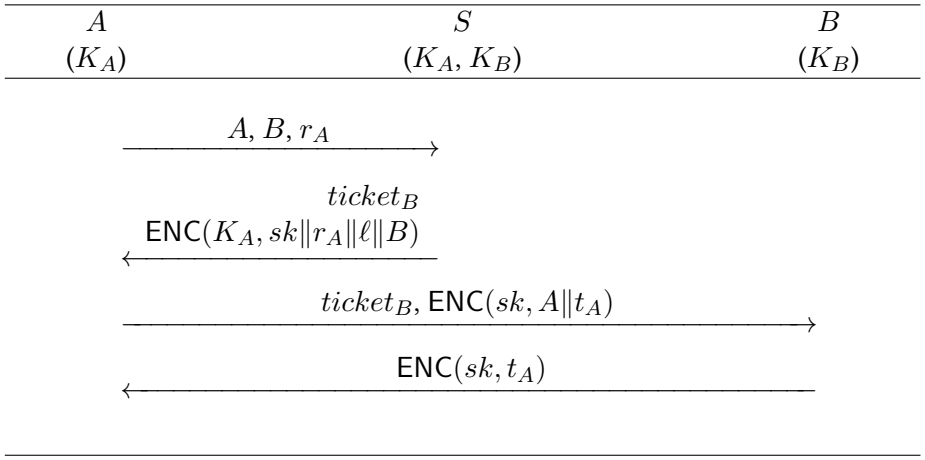


Figure 1.6 – The Kerberos v5 protocol. $ticket_B$ corresponds to $\mathsf{ENC}(K_B, sk\|A\|\ell)$.

Smart cards. The SCP02 [Glo18] and SCP03 [Glo14] protocols are used to establish a secure channel between a server or reader and a remote smart card (e.g., UICC/SIM card). They are built on a single core function (respectively DES/3DES and AES). These protocols are widely used to manage the content of SIM cards (update of secret parameters, install of applets). Their security is also based on static symmetric keys. Although the static master keys can be updated (the new master keys are protected with session keys computed with the current master keys), this depends on the overall security policy of the service provider that deploys the cards, and is not part of the SCP02 and SCP03 protocols per se. In addition, the session keys output by these two protocols depend on a monotonically increasing counter respectively 15 and 24-bit long which limits the number of runs. Furthermore, only the server can initiate a protocol run. As we will see in Chapter 4, SCP02 enjoys more crippling features. Figure 1.9 depicts the key establishment phase in SCP02.

The message flow in SCP03 is similar to that of SCP02. The main differences are the symmetric functions used to compute the messages and the session keys, and the fact that the sequence counter in SCP03 is incremented at the beginning of a protocol run, whereas in SCP02, it is incremented after a successful run.

Radio identification (RFID). In the RFID field, several protocols based on a shared master key propose to update the master key throughout the protocol run. For instance, Le, Burmester, and de Medeiros [LBM07] propose O-FRAKE that aims at authenticating a tag to a server, and at computing a session key in order to establish a secure channel (see Figure 1.10).

Only the server can initiate a protocol run. In order to ensure forward security, the master key is updated throughout the protocol run. To deal with the possible desynchronisation between

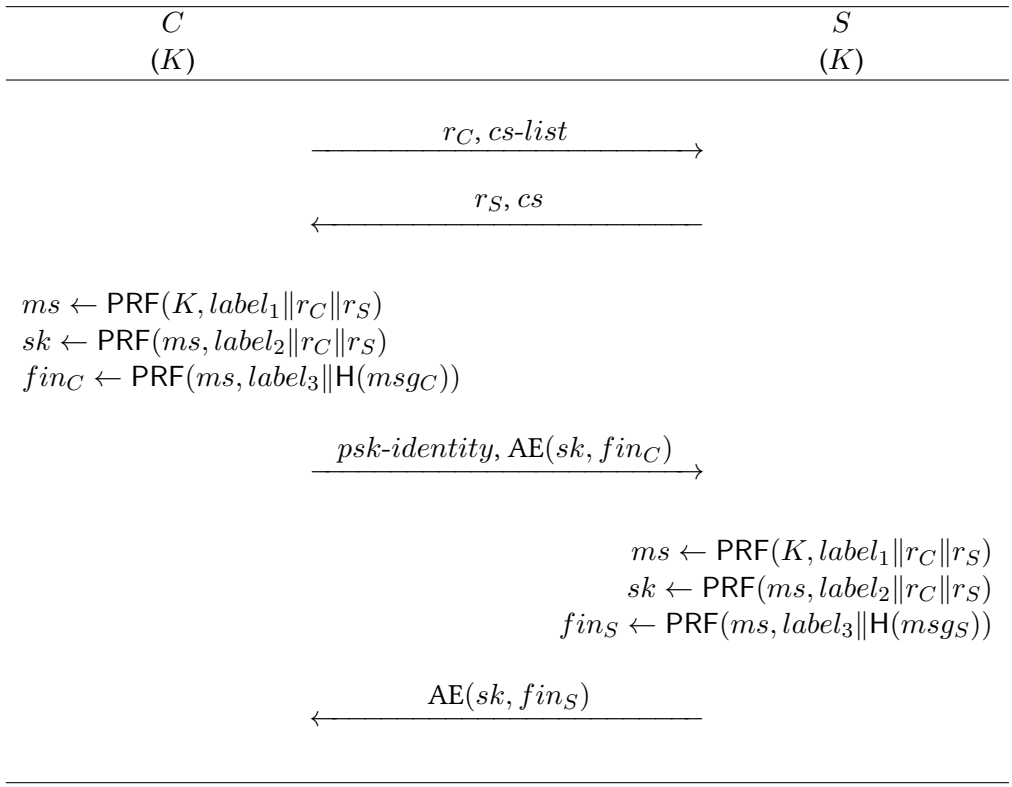


Figure 1.7 – The key establishment in TLS 1.2 protocol in PSK mode. $cs\text{-}list$ denotes the list of cipher suites proposed by the client (C). msg_C and msg_S correspond to the transcript of messages sent and received so far respectively by the client and the server. Abusing the notation, AE denotes that the final messages are encrypted and MAC-ed with the session key (sk).

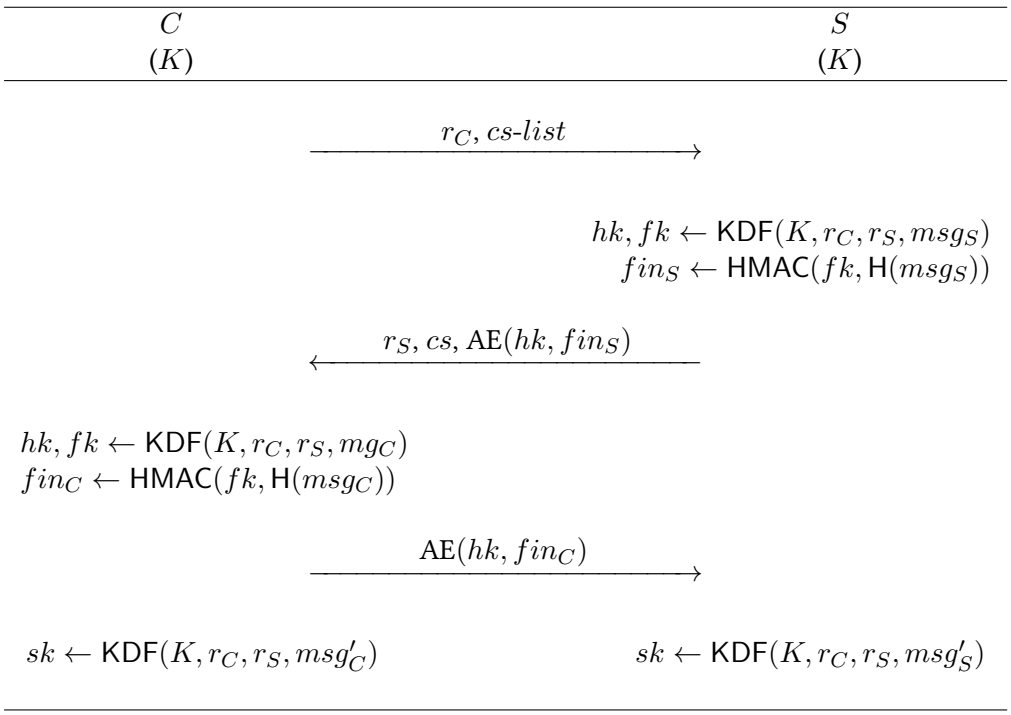


Figure 1.8 – The key establishment in TLS 1.3 protocol in PSK mode. KDF denotes the computation through intermediary steps of several keys (hk, fk, sk) with the pre-shared key K, and the HKDF function [KE10]. The underlying hash function for HMAC and HKDF depends on the chosen ciphersuite cs. msg_C, msg'_C and msg_S, msg'_S correspond to the transcript of messages received and computed so far respectively by the client (C) and the server (S).

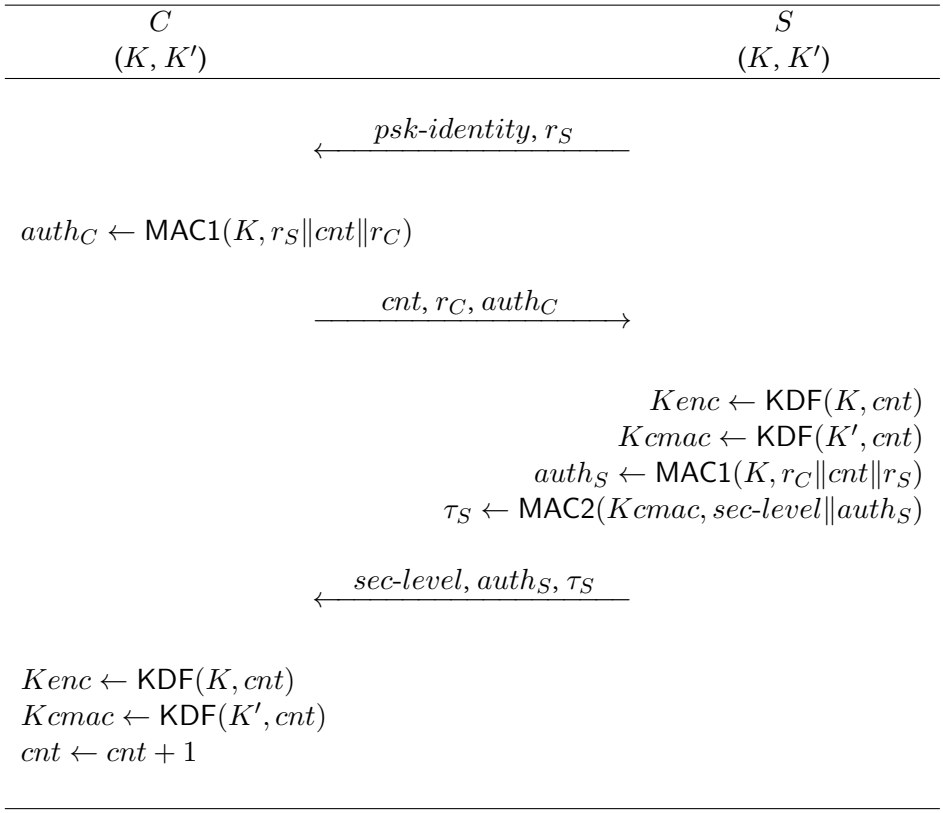


Figure 1.9 – The key establishment in SCP02 protocol. The KDF, MAC1, and MAC2 functions are based on DES and 3DES.

the reader and the tag, the server keeps two consecutive values of the key: the current and the previous one. The server is always at most one step ahead of the tag. Therefore, if the tag does not update its master key, the server is able to catch up. But this implies that, in case of desynchronisation, the server computes the session key from the updated master key, whereas the tag still stores the previous value. Hence, an adversary that corrupts that tag can compute the previous session key with respect to the server (i.e., the adversary breaks the forward secrecy). In fact, since the server always keeps the previous value of the master key, together with the current one, the scheme is intrinsically insecure in strong security models (i.e., models that allow the adversary to corrupt any of the partners, once the targeted party has accepted).

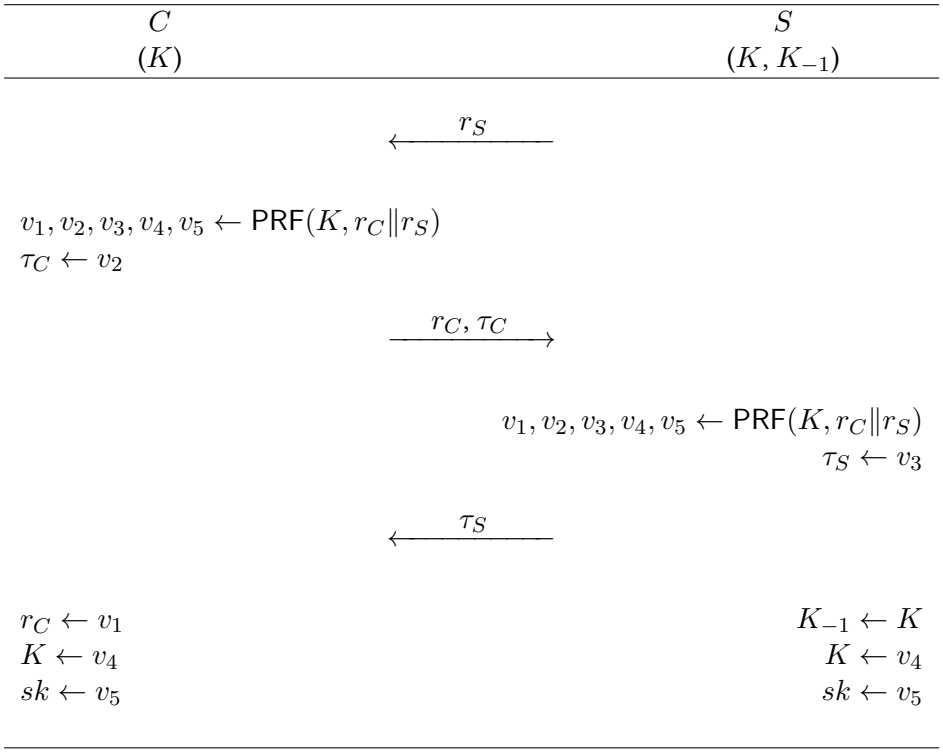


Figure 1.10 – The O-FRAKE protocol. Once the server (S) sends τ_S, the master key K is updated. The server keeps the previous master key K_{-1}, in case the tag (C) does not receive τ_S, and does not update its own version of the master key.

Wireless sensor networks (WSN). The area of WSN requires also lightweight, efficient AKE protocols, and has been subject to numerous proposals. One can cite in particular the following ones. Perrig, Szewczyk, Tygar, Wen and Culler [PST+02] propose the SPINS protocol, based on one encryption function. This protocol implies a central server. This server generates and sends a fresh session key to pairwise sensors, encrypted with each sensor's master key, which is static and set before the sensors are deployed on the field (see Figure 1.11). SPINS includes a procedure to broadcast authenticated messages. The first key k_0 is transmitted to each node authenticated with the node's master key. Each subsequent authentication key belongs to a one-way chain: $k_i = \mathsf{H}(k_{i+1})$, $i \geq 0$. The broadcast procedure is the following. At time t_i, each node knowns key k_i. At time t_{i+1}, the server broadcasts a message m_{i+1} MAC-ed with key k_{i+1}. Then, at time $t_{i+1} + \Delta < t_{i+2}$, the server discloses k_{i+1}. Each node can verify that

$k_i = \mathsf{H}(k_{i+1})$, and accept message m_{i+1}. Key k_{i+1} expires at time t_{i+2}. This protocol implies a synchronised clock between the server and all the nodes, otherwise an attacker can compute valid messages with key k_{i+1} although the key is expired.

In the same field, Park and Shin propose LiSP [PS04]. It is also based on a central server which shares a different master key with each sensor. The initial session key is sent to each sensor encrypted with its static master key, and each subsequent session key is encrypted (with no MAC) with the previous one. The master key must not be deleted because, as in SPINS, the session keys belong to a chain computed with a one-way function H. Each key is computed as the image of the next one: $k_i = \mathsf{H}(k_{i+1})$. When a sensor receives k_{i+1} (encrypted with k_i), it checks if k_{i+1} is a pre-image of k_i, and then replaces k_i with k_{i+1}. Therefore, the number of session key update is limited by the length of the hash chain, and the master key is used to periodically initialise a new chain. The protocol implies a (loosely) synchronised clock between all the nodes and the central server in order to trigger the session key update but also to avoid replay of messages (in particular the messages sent by the server to initiate a new session key chain).

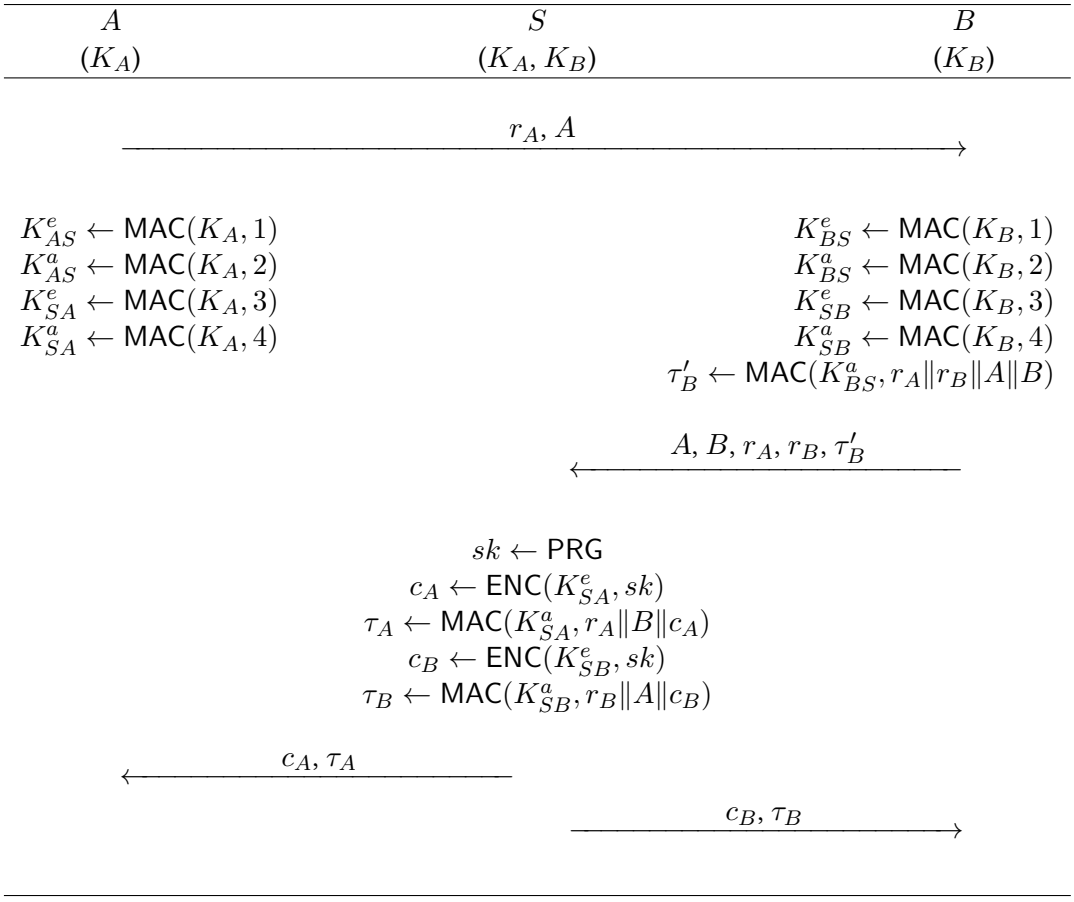


Figure 1.11 – The key establishment in SPINS protocol

Smart home. Regarding the area of the smart home, several protocols make use of static master keys. In order to share a master key, the master device in ZigBee [Zig14] merely sends the key in clear to the joining device. Hence passively eavesdropping on this phase is enough to undermine the security of a ZigBee network.

The WPA2 protocol [IEE04] used in WiFi communications is a client-server type protocol based on a shared static master key. From two 48-bit nonces (r_C, r_S) exchanged by the client and the server, several session keys are derived (see Figure 1.12). They are used to protect subsequent handshake messages (ek, ak), and application data (sk). WPA2 has been subject to several attacks by Vanhoef and Piessens [VP17; VP18] that allows replaying and decrypting encrypted messages, based on the ability to compel a device to reuse a session key whereas encryption is done in CTR mode.

In Z-Wave S0 [Sil18] the master key is sent encrypted with a known (all zero) key. Hence eavesdropping on the key exchange phase undermines the security of the network.

In Z-Wave S2 [Sig16] a static variant of the scheme ECDH scheme is used. Whereas the master device generates a fresh ECDH share, the joining device uses always the same share. The latter is printed on the joining device in the form of a QR code, or an hexadecimal string. Authentication is done by comparing the ECDH share received by the master device and the value printed on the joining device's sticker. The secret key output by the ECDH exchange encrypts the master key (known also to all other devices members of the same Z-Wave network), and the ciphertext is sent to the joining device. Eventually, this master key is used to protect the messages exchanged between the devices.

We observe that the authentication of the joining device relies upon the integrity of its sticker. If an attacker succeeds in replacing the sticker with one of her choice (corresponding to an ECDH public value chosen by her), then she can make a proper ECDH key exchange with the master device, and consequently receive the network's master key. The legitimate user deems the authentication is valid since the received ECDH value is equal to the (fake) sticker on the joining device. Even if the legitimate user repeats the pairing procedure with his new device (which was unable to receive the master key during the Man-in-the-Middle attack), the same master key is sent to the latter. Hence, this scenario allows the attacker to decrypt all subsequent application messages exchanged between the legitimate devices, and to transmit valid messages.

As one can see, on the one end, Z-Wave S2 tries to enhance the security through asymmetric cryptography, but, on the other hand, the protocol does not make use of all the properties enabled by the public key schemes due to the constrained resources of the Z-Wave devices (i.e., use of a static ECDH share, no forward secrecy, no proper entity authentication).

Tierney [Tie18] has also shown how to compel a Z-Wave device to downgrade from version S2 to S0, which impairs the security.

Mobile telephony. In mobile telephony, the Universal Mobile Telecommunications System (UMTS)[3rda] and Long Term Evolution (LTE) [3rdb] system are based on static master keys shared between the user's SIM card and a back-end server. From these static master keys (and random values exchanged throughout the protocol run), all the session keys are computed. The number of sessions is bounded by a 48-bit counter.

Controller area networks (CAN). In the field of the automotive security, several protocols have been proposed based on symmetric-key functions (e.g., [VHSV11; GMVV12; BSNRN14; NR16; VBMP17]). They aim at authenticating the messages exchanged by vehicle control units in order to command sensitive elements of the vehicle (e.g., brake, steering). For instance, among the latest proposals, Radu and Garcia [RG16] describe LeiA where each session key is computed from the shared, static master key, and a (synchronised) counter. Rather than an interactive key agreement protocol, LeiA includes an extra procedure in order for the parties to resynchronise if needed.

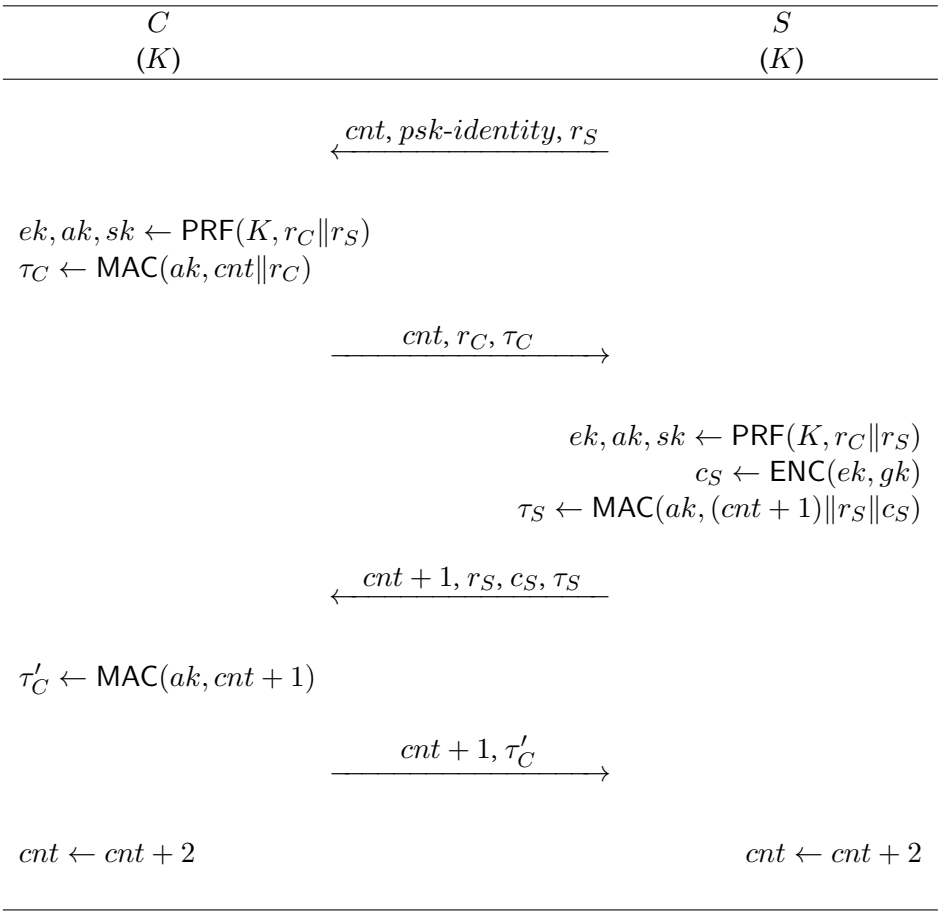


Figure 1.12 – The key establishment in WPA2 protocol. cnt is a global counter related to the master key K. gk is a group key sent by the server.

Internet of Things. Regarding the IoT field, the following protocols are widely deployed or strongly promoted. They all build their security on symmetric-key functions, and make use of a static and unique (per end-device) root key shared between the end-device and the back-end network.

Sigfox [Sig17b; Sig17a] is a communication and security protocol dedicated to long-range, low-power devices (LPWAN). Each end-device connects to a central server. The data is protected with the same static symmetric keys, used during the whole end-device's lifespan (Sigfox proposes no key exchange procedure).

LoRaWAN 1.0 [LoR18a] and 1.1 [Sor17] are two versions of a protocol that aims at securing the communication between end-devices with low computational and energy resources (e.g., sensor, actuator) and a back-end server. The session keys are derived from the end-device's root keys. As all the aforementioned protocols, these do not provide forward secrecy. In addition, due to the short size of the parameters (pseudo-random values or counters), the number of protocol runs is bounded. Furthermore, only the end-device (but not the central server) can initiate a session. In Chapter 3, we look in depth at these protocols and show that they suffer from several weaknesses that lead to likely practical attacks.

Contrary to the previous technologies, Narrowband IoT (NB-IoT), enhanced Machine-Type Communication (eMTC), Extended Coverage GSM IoT (EC-GSM-IoT) are cellular technologies. eMTC provides enhancements to the Long Term Evolution (LTE/4G) technology for machine type communications. NB-IoT is also based on LTE, whereas EC-GSM-IoT is based on GSM/EDGE technologies and dedicated to constrained end-devices. These technologies aims at decreasing the end-device complexity (hence its cost), power consumption, extending autonomy, and increasing coverage [GSM16]. The security of all these systems relies on the underlying technology (GSM, EDGE, LTE), hence on a static symmetric root key known to a central authority, likely the telecom operator. They inherit the intrinsic security limitations of the symmetric-key schemes they are built on.

1.3.2 The Importance of Being Proved

Despite the great value of the paradigm of provable security, only a few of the protocols described in Section 1.3.1 have been formally proved. Furthermore, the security models used do not grant the adversary the same powers, hence the resulting proof are not equally meaningful.

Thus, in their proof for UMTS and LTE, Alt et al. [AFM+16] do not allow any server corruption. Furthermore, they allow the adversary to get either one or the other, but not both master keys needed to compute the session keys.

Radu and Garcia [RG16] prove their protocol secure in the Dolev-Yao model [DY81]. They do not allow any party corruption, and their protocol does not provide forward secrecy either.

In the security model of Dousti and Jalili [DJ14], the adversary has access to queries that allow registering new parties, sending data to some party, corrupting a party, and getting the session state (which includes the session keys, and public values such as the session identifier – exchanged in clear throughout the protocol run –, the partner identifier, the party's role, and the time when the session is initiated). More significantly, Dousti and Jalili define security based on the session key indistinguishability, but do not consider entity authentication. In addition, their notion of "freshness" (used to define the characteristics of the security experiment) demands that the targeted party have an *actual* partner (rather than demanding only that the targeted party accept with some *intended* partner). Furthermore, corruption of the targeted party or its intended partner is not allowed before both have computed the session keys, in contrast to stronger models where the adversary can corrupt either of these two parties as soon as the

targeted party accepts. More important, it is assumed a perfect time synchronisation, and a simultaneous and automatic update of the master keys at the two intended partners (hence the adversary is forbidden from attempting to desynchronise the parties by interacting with them). The update is made at regular time intervals. We observe that, in their experiment, such a duration is not defined in relation to the duration of a protocol run. In particular, if the time limit exceeds the duration of a session, then the adversary can trivially win the security experiment (the adversary issues a Corrupt-query after the session keys are computed but before the master key is updated, and recomputes the session keys).

In the model used by Van Le et al. [LBM07], corrupting the server is not allowed, and corruption of the tag is possible only once both parties have accepted.

Despite the fact that these protocols aim at being fully functional on constrained devices, they require a careful cryptographic analysis against strong threat models. Moreover, some of these protocols involve more than two parties. Hence, their security model must also capture the interleaved operations that result from this complexity of interactions.

1.3.3 Selective Summary

As a summary, Table 1.2 compares the protocols presented in Section 1.3.1 with respect to several significant criteria.

"Security". A check-mark (✓) indicates the existence of a security proof, and is followed by the corresponding reference. On the contrary, if references follow a cross-mark (✗), they correspond to attacks against the protocol. Note that each proof may correspond to different settings (computational, symbolic), and security models. A check-mark followed by a cross-mark indicates the existence of both a security proof and attacks.[2] This is the case for UMTS, LTE, and WPA2.

Regarding SCP03, despite the lack of a security proof for the protocol, Sabt and Traoré [ST16] have provided a security proof of the underlying confidentiality and integrity algorithms.

Despite the proof provided by Alt, Fouque, Macario-Rat, Onete and Richard [AFM+16] on the key establishment protocol of UMTS and LTE, there exist attacks against UMTS (e.g., [MW04; ZF05; ASS09]) and LTE (e.g., [TM12; HC14; RKHP19]). These attacks are either based on stronger powers granted to the adversary than that of Alt et al., or exploit the possible interworking between different technologies (e.g., UMTS and Global System for Mobile Communication – GSM), or target the (post-accept) secure channel.

"PFS". It indicates if the protocol guarantees forward secrecy. We stress that, in that case, we do consider a "strong" security model to assess if that property is indeed provided by the considered protocol. Indeed, forward secrecy is of paramount importance in regards to the security level we want to meet. Hence we use the same criterion to compare the protocols (concretely, we consider an adversary that can corrupt the targeted party and its intended partner as soon as the targeted party accepts).

"#msg". It gives the number of messages exchanged during a correct protocol run.

"#ses". It gives the number of possible sessions. The symbol "∞" indicates that the protocol allows virtually an unlimited number of runs.

Regarding WPA2 [IEE04], the session keys computation is based on two 48-bit pseudo-random values. Hence our choice of $\sqrt{2^{96}}$ for the number of sessions before the session keys repeat (in

[2] We do not claim to provide an exhaustive list of references.

presence of a passive adversary).

Likewise, despite the fact that LoRaWAN 1.0 [LoR18a] does not limit the number of sessions, collisions occur on the session keys with high probability after $\sqrt{2^{40}}$ key exchanges (in presence of a passive adversary).

Regarding LoRaWAN 1.1 [Sor17], the parameter j is implementation-dependent.[3]

"I/R". It indicates if any (type of) party can be initiator or responder of a session. A check-mark denotes the affirmative whereas a cross-mark denotes the opposite. For instance, AKEP1 allows any party to initiate a session. On the contrary, only the server can initiate an SCP02 session (and only a smart card can respond to it).

"P2P". It indicates if the key establishment is point-to-point between the two intended partners. A cross-mark denotes that a third party (e.g., key server) is required.

"Sync". It indicates if an extra procedure or functionality (in addition to the key establishment procedure) is needed in order for the parties to guarantee some form of synchronisation. A "yes" answer implies a supplementary burden (e.g., in terms of computation, implementation surface, energy).

[3] According to us, it is likely that $j = 1$, as explained in Chapter 3.

Table 1.2 – Comparison of several key exchange protocols (all but one) in the symmetric-key setting

Protocol	Security	PFS	#msg	#ses	I/R	P2P	Sync
AKEP1, AKEP2 [BR94]	✓ [BR94]	✗	3	∞	✓	✓	no
ISO/IEC 11770-2 [Int08]	✗ [CC04] [MS08] [CM08] [CH14]	✗	1-5	∞	✓	✓ (mech. 1-6) ✗ (mech. 7-13)	no
FORSAKES [DJ14]	✓	✓	3	∞	✓	✓	yes
Kerberos v5 [NYHR05]	✓ [BCJ+06] [BK07] [BJST08]	✗	4	∞	✗	✗	yes
TLS 1.2 PSK [ET05]	✓ [LSY+14]	✗	4	∞	✗	✓	no
TLS 1.3 PSK [Res18]	✓ [CHH+17] [DFK+17] [PS18] [ABF+19]	✗	3	∞	✗	✓	no
SCP02 [Glo18]	✗ [ST16] Chapter 4	✗	3	2^{15} (per master key)	✗	✓	no
SCP03 [Glo14]	✓ [ST16]	✗	3	2^{24} (per master key)	✗	✓	no
O-FRAKE [LBM07]	✓ [LBM07]	✗	3	∞	✗	✓	no
SPINS [PST+02]	✗	✗	4	∞	✓	✗	yes
LiSP [PS04]	✗	✗	1	∞	✓	✗	no
ZigBee [Zig14]	✗	✗	4	∞	✗	✗	no
WPA2 [IEE04]	✓ ✗ [HSD+05] [VP16] [VP17] [VP18]	✗	4	$< 2^{48}$	✗	✓	no
Z-Wave S0 [Sil18]	✗	✗	8	∞	✗	✗	no
Z-Wave S2 [Sig16]	✗ [Tie18] Section 1.3.1	✗	16	∞	✗	✗	no
UMTS [3rda]	✓ ✗ [AFM+16] [MW04] [ZF05] [ASS09]	✗	2×7	2^{48}	✗	✗	yes
LTE [3rdb]	✓ ✗ [AFM+16] [TM12] [HC14] [RKHP19]	✗	2×7	2^{48}	✗	✗	yes
LeiA [RG16]	✓ [RG16]	✗	0	2^{56}	✓	✓	yes
LoRaWAN 1.0 [LoR18a]	✗ Chapter 3	✗	2	$< 2^{20}$	✗	✓	no
LoRaWAN 1.1 [Sor17]	✓ Chapter 5	✗	≥ 4	$\min(j2^{16}, 2^{24})$	✗	✓	no

1.4 Contributions of this Thesis

Our contributions are threefold:

- We analyse existing symmetric-key protocols (including widely deployed ones) and show that they suffer from weaknesses that allow implementing likely practical attacks. As a concrete demonstration we provide the results of a successful attack done in an experimental setting against smart cards implementing one of the analysed protocols.

- We devise suitable security models that we subsequently use to analyse two symmetric-key protocols, including one that we have conceived.

- We present two- and three-party authenticated key exchange protocols in the symmetric-key setting that provide stronger security properties than the existing protocols. The additional properties and functionality that we obtain include the essential forward secrecy property, and session resumption. The latter functionality is particularly advantageous for low-resource end-devices with constrained capabilities in terms of communication and computation.

The papers accepted in conferences (and, for most of them, already presented) during this thesis are the following:

[ACF19] Gildas Avoine, Sébastien Canard, and Loïc Ferreira. *IoT-Friendly AKE: Forward Secrecy and Session Resumption Meet Symmetric-Key Cryptography*. In: *ESORICS 2019, Part II*. Ed. by Kazue Sako, Steve Schneider, and Peter Y. A. Ryan. Vol. 11736. LNCS. Springer, Heidelberg, Sept. 2019, pp. 463–483.

[ACF20] G. Avoine, S. Canard, and L. Ferreira. *Symmetric-key Authenticated Key Exchange (SAKE) with Perfect Forward Secrecy*. In: *CT-RSA*. (To appear). 2020.

[AF18a] Gildas Avoine and Loïc Ferreira. "Attacking GlobalPlatform SCP02-compliant Smart Cards Using a Padding Oracle Attack". In: *IACR TCHES* 2018.2 (2018). `https://tches.iacr.org/index.php/TCHES/article/view/878`, pp. 149–170. ISSN: 2569-2925.

[AF18b] Gildas Avoine and Loïc Ferreira. *Rescuing LoRaWAN 1.0*. In: *FC 2018*. Ed. by Sarah Meiklejohn and Kazue Sako. Vol. 10957. LNCS. Springer, Heidelberg, 2018, pp. 253–271.

[CF19] Sébastien Canard and Loïc Ferreira. *Extended 3-Party ACCE and Application to LoRaWAN 1.1*. In: *AFRICACRYPT 19*. Ed. by Johannes Buchmann, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 11627. LNCS. Springer, Heidelberg, July 2019, pp. 21–38.

1.4.1 Cryptographic Analysis of Existing Protocols [AF18b; AF18a; CF19]

As mentioned in Section 1.1, existing symmetric-key protocols do not provide as strong security properties as protocols based on asymmetric schemes. We enlighten this fact through the analysis of two protocols currently deployed worldwide, and implemented in a numerous amount of devices.

Analysis of LoRaWAN. The first protocol is LoRaWAN. This IoT protocol dedicated to long-range, low-power wide area networks (LPWAN) aims at securing communications between connected objects and a back-end network. These objects are intended to provide various types of services from telemetry measurement, remote device activation, site surveillance and intrusion detection, geolocation of valuable assets, up to management of networks that deliver sensitive resources (e.g., water, energy).

We provide an extensive analysis of the protocol, and show that it suffers from several weaknesses. We introduce several attacks, including likely practical ones, that breach the network availability, data integrity, and data confidentiality, and present two different attack scenarios. Based on the inner weaknesses of the protocol, these attacks do not lean on potential implementation or hardware bugs. Likewise they do not entail a physical access to the targeted equipment (for one of the attack settings), and are independent from the means used to physically protect secret parameters. Finally we propose practical recommendations aiming at thwarting the attacks, while at the same time being compliant with the specification, and keeping the interoperability between patched and unmodified equipment.

The version 1.1 aims at correcting the flaws that impair the previous version. We describe several weaknesses that still affect this latest version, and present recommendations in order to mitigate these flaws.

LoRa Alliance, in charge of developing and promoting the LoRaWAN specification, has been informed of the attacks described in our paper [AF18b]. A document recommending changes to be implemented in LoRaWAN 1.0 has been published [LoR18b], and the subsequent version LoRaWAN 1.1 incorporates also some suggestions yielded by our analysis.

Analysis of SCP02. The second security protocol is SCP02. Implemented in particular in smart cards, this protocol is deployed by transport companies, mobile network operators (UICC/SIM cards), and in the banking world to securely transmit sensitive data.

We show how to perform a padding oracle attack against the SCP02 protocol. This attack allows to efficiently retrieve data protected within the secure channel. Furthermore, we have implemented the attack in an experimental setting, and we present the results of our experiments done with 10 models of smart cards produced by six different card manufacturers. This shows that the attack is fully practical in our experimental setting. To the best of our knowledge, this is the first successful attack against the SCP02 protocol.

GlobalPlatform has been informed of our analysis. While our paper [AF18a] was under submission, GlobalPlatform has decided to deprecate the SCP02 protocol. Moreover, the manufacturers of the tested smart cards have received the technical details of our work prior to being presented at the conference (between October 2017 and March 2018).

These results have been presented in the following international conferences: Financial Cryptography and Data Security (FC, 2018), Cryptographic Hardware and Embedded Systems (CHES, 2018) and Africacrypt (2019). They are detailed in Chapters 3 and 4.

1.4.2 Design of Security Models [CF19; ACF19]

In order to use a provable security approach, we define two security models that are subsequently applied to analyse the security of two protocols, among which one that we have devised.

The 3-AKE security model. The first model, that we call 3-AKE, aims at analysing the security of 3-party authenticated key exchange protocols (AKE). This model captures in particular

the forward secrecy property. In Chapter 7, we present a generic 3-party AKE and, based on it, a concrete instantiation, both in the symmetric-key setting. We use the 3-AKE model to formally prove the security of these two schemes.

The 3-ACCE security model. The second security model is based on the ACCE paradigm devised by Jager, Kohlar, Schäge and Schwenk [JKSS11] to cope with the difficulty in proving the security of a protocol based on the indistinguishability of the session keys, when the latter are used during the key exchange phase. Our model allows analysing 3-party protocols which goal is to set up secure channels, hence its name 3-ACCE.

First, we use this framework to prove the security of a generic LoRaWAN-like protocol. Then we present an adapted version of LoRaWAN 1.1 with stronger security properties, modified in order to mitigate the vulnerabilities described in Chapter 3. Applying the first result, we formally prove the security of this protocol in our model, and describe how to concretely instantiate it.

These results have been presented during the following international conferences: Africacrypt (2019) and European Symposium on Research in Computer Security (ESORICS, 2019). They are described in greater detail in Chapter 5.

1.4.3 Design of Forward Secret Symmetric-key Protocols [ACF20; ACF19]

As witnessed by the protocols described in Section 1.3, achieving both efficiency and security is a non-trivial task. Contrary to all the aforementioned protocols, we aim at describing lightweight protocols that are simple to deploy, and do not require complex practical requirements (e.g., time synchronisation between principals). Consequently, we propose two authenticated key exchange protocols that are built on symmetric-key functions solely, and ensure forward secrecy. These protocols do not make trade-off between efficiency and security, and are analysed in a strong security model.

The SAKE protocol. First, we describe a two-party symmetric-key protocol that we call SAKE. Based on a shrewd synchronisation mechanism and a key evolving scheme, the protocol ensures forward secrecy.

The 3-AKE protocol. Then, we describe a generic 3-party authenticated key exchange protocol dedicated to IoT, that we call 3-AKE. It involves a (wireless) end-device, a server, and a trusted third party used as authentication and key server. Solely based on symmetric-key functions (regarding the computations done between the end-device and the back-end network), this protocol guarantees also forward secrecy. In addition, it enables session resumption without impairing security (in particular, forward secrecy is maintained). This allows saving communication and computation cost, and is advantageous for low-resource end-devices.

We present a concrete instantiation of the 3-AKE protocol based on the two-party protocol SAKE.

The 3-party key exchange protocol can be applied in a real-case IoT deployment (i.e., involving numerous end-devices and servers) such that the latter inherits from the security properties of the protocol. This results in the ability for a (mobile) end-device to securely switch from one server to another back and forth at a reduced (communication and computation) cost, without compromising the sessions established with other servers.

These results have been presented at the European Symposium on Research in Computer Security (ESORICS, 2019). They are detailed in Chapters 6 and 7.

2 Preliminaries and Definitions

In this chapter we present the basic cryptographic definitions and building blocks we need in order to elaborate more complex mechanisms, and to formally analyse the security of protocols in the subsequent chapters.

Contents

2.1 Notation

7F	A byte (equal to the hexadecimal value 0x7F)
00^i	Byte string made of i times the byte 00
$var\ (4)$	A variable var which is 4-byte long
$\|x\|$	Absolute value of x
$x \| y$	Concatenation of bytes or string of bytes x and y
$b_i \| b_{i+1}$	Concatenation of bits b_i and b_{i+1}
$x \xleftarrow{\$} E$	Value x chosen uniformly at random in the set E
$y \leftarrow F(x)$	Variable y to which the value $F(x)$ is attributed
$\{0,1\}^*$	Set of all binary strings
$\{0,1\}^n$	Set of all n-bit strings
ENC^{-1}	Inverse function of function ENC
$\Pr[A]$	Probability of event A

2.2 Preliminaries

In this section, we recall the definitions of the main security notions we use in Chapters 5, 6 and 7. The security definition of a pseudo-random function and pseudo-random permutation is taken from Bellare, Desai, Jokipii, and Rogaway [BDJR97], and that of a MAC strongly unforgeable under chosen-message attacks from Bellare and Namprempre [BN08]. We take the definition of a stateful authenticated encryption scheme (sAE) from Bellare, Kohno and Namprempre [BKN02] using the notation of Krawczyk, Paterson and Wee [KPW13]. We recall also the definition of matching conversations initially proposed by Bellare and Rogaway [BR94], and modified by Jager, Kohlar, Schäge, and Schwenk [JKSS11].

2.2.1 Matching Conversations

Let $T_{i,s}$ be the sequence of all (valid) messages sent and received by an instance π_i^s in chronological order. Let $\ell(T_{i,s})$ be the number of messages in $T_{i,s}$. For two transcripts $T_{i,s}$ and $T_{j,t}$, we say that $T_{i,s}$ is a prefix of $T_{j,t}$ if $\ell(T_{i,s}) \geq 1$, and the messages in $T_{i,s}$ are identical to the first $\ell(T_{i,s})$ messages of $T_{j,t}$.

Definition 2.1 (Matching Conversations)**.** *We say that π_i^s has a* matching conversation *to π_j^t, if*

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

Remark. Defining matching conversations as per Definition 2.1 means that we use a post-specified session identifier equal to the first but not necessarily all messages of the protocol analysed. As explained by Jager et al., this asymmetry is necessary, due to the fact that protocol messages are sent sequentially. Therefore one of the two parties has to accept without knowing if the other party has received all messages. Indeed an adversary is always able to drop the last message in a protocol run.

2.2.2 Pseudo-random Function

Let k be the security parameter. A *pseudo-random function* (PRF) F is a deterministic algorithm which given a key $K \in \{0,1\}^k$ and a bit string $x \in \{0,1\}^*$ outputs a string $y = F(K,x) \in$

$\{0,1\}^\gamma$ (with γ being polynomial in k). 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 $K \xleftarrow{\$} \{0,1\}^k$, $G \xleftarrow{\$} Func$, 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 its 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 2.2 (Secure PRF)**.** *A function F: $\{0,1\}^k \times \{0,1\}^* \to \{0,1\}^\gamma$ is said to be a* secure pseudo-random function (PRF) *if, for all probabilistic polynomial time adversary $\mathcal{A}$, $\mathsf{adv}_F^{\mathsf{prf}}(\mathcal{A})$ is a negligible function in k.*

2.2.3 Pseudo-random Permutation

A *pseudo-random permutation* (PRP) F is a deterministic algorithm which given a key $K \in \{0,1\}^k$, where k is the security parameter, 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 k). 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 $K \xleftarrow{\$} \{0,1\}^k$, $G \xleftarrow{\$} Perm$, $b \xleftarrow{\$} \{0,1\}$ 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 its 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 2.3 (Secure PRP)**.** *A function F: $\{0,1\}^k \times \{0,1\}^\gamma \to \{0,1\}^\gamma$ is said to be a* secure pseudo-random permutation (PRP) *if, for all probabilistic polynomial time adversary $\mathcal{A}$, $\mathsf{adv}_F^{\mathsf{prp}}(\mathcal{A})$ is a negligible function in k.*

2.2.4 Message Authentication Code

A *message authentication code* (MAC) consists of three algorithms $\mathsf{Tag} = (\mathsf{Tag.Gen}, \mathsf{Tag.MAC}, \mathsf{Tag.Vrf})$. The probabilistic algorithm $\mathsf{Tag.Gen}$ samples a key K from the set $\{0,1\}^k$: $K \leftarrow \mathsf{Tag.Gen}()$. The algorithm $\mathsf{Tag.MAC}$ takes as input a key $K \in \{0,1\}^k$ and a message $m \in \{0,1\}^*$, and returns a tag $\tau \in \{0,1\}^\gamma$ (with γ being polynomial in k). The verification algorithm $\mathsf{Tag.Vrf}$ takes as input the key K, a message m, and a candidate tag τ for m. It outputs `true` if τ is a valid tag on message m with respect to K. Otherwise, it returns `false`.

We require that a MAC be correct. That is, for all $K \in \{0,1\}^k$ and for all $m \in \{0,1\}^*$, we have that $\mathsf{Tag.Vrf}(K, m, \mathsf{Tag.MAC}(K, m)) = \mathtt{true}$.

The notion of *strong unforgeability under chosen-message attacks* (SUF-CMA) for a MAC $\mathsf{Tag} = (\mathsf{Tag.Gen}, \mathsf{Tag.MAC}, \mathsf{Tag.Vrf})$ is defined with the following experiment between a challenger and an adversary $\mathcal{A}$:

1. The challenger samples $K \leftarrow \mathsf{Tag.Gen}()$, and sets $S \leftarrow \emptyset$.

2. The adversary may adaptively query values m to the challenger. The challenger replies to each query with $\tau = \mathsf{Tag.MAC}(K, m)$ and records (m, τ): $S \leftarrow S \cup \{(m, \tau)\}$.
   In addition, the adversary may adaptively query values (m', τ') to the challenger. The challenger replies to each query with $\mathsf{Tag.Vrf}(K, m', \tau')$.

3. Finally, the adversary sends (m^*, τ^*) to the challenger.

The adversary's advantage is defined as

$$\mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Tag}}(\mathcal{A}) = \Pr[\mathsf{Tag.Vrf}(K, m^*, \tau^*) = \mathtt{true} \wedge (m^*, \tau^*) \notin S].$$

Definition 2.4 (SUF-CMA)**.** *A message authentication code* $\mathsf{Tag} = (\mathsf{Tag.Gen}, \mathsf{Tag.MAC}, \mathsf{Tag.Vrf})$ *with* $\mathsf{Tag.MAC}$: $\{0,1\}^k \times \{0,1\}^* \to \{0,1\}^\gamma$ *is said to be* strongly unforgeable under chosen-message attacks (SUF-CMA) *if, for all probabilistic polynomial time adversary* $\mathcal{A}$, $\mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Tag}}(\mathcal{A})$ *is a negligible function in* k.

2.2.5 Stateful Authenticated Encryption

We present two flavours for the definition of a stateful authenticated encryption scheme. The first one guarantees that the encryption scheme "hides" the plaintext such that it is not possible to distinguish the ciphertext corresponding to a given plaintext, and the ciphertext corresponding to a random value (real-or-random indistinguishability or RoR). The second guarantees that it is not possible to distinguish two different plaintexts based on their corresponding ciphertexts (left-or-right indistinguishability or LoR). Bellare et al. [BDJR97] have shown that both notions are equivalent. We use either definition to match with the specific security experiment considered in the proofs given in Chapters 5 and 7.

The difference between both definitions lies in the description of the Encrypt oracle. Figures 2.1 and 2.2 describe respectively the RoR and LoR variants when the encryption function is a stateful authenticated encryption scheme which we define below.

A *stateful authenticated encryption scheme* (sAE) consists of the following four algorithms $\mathsf{StAE} = (\mathsf{StAE.Gen}, \mathsf{StAE.Init}, \mathsf{StAE.Enc}, \mathsf{StAE.Dec})$. The probabilistic algorithm $\mathsf{StAE.Gen}$ samples a key K from the set $\{0,1\}^k$: $K \leftarrow \mathsf{StAE.Gen}()$. The deterministic algorithm $\mathsf{StAE.Init}$ initialises two states st_e and st_d respectively for encryption and decryption: $(st_e, st_d) \leftarrow \mathsf{StAE.Init}()$. The encryption algorithm, given as $(C, st'_e) \leftarrow \mathsf{StAE.Enc}(K, H, M, st_e)$, takes as input a secret key $K \in \{0,1\}^k$, 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 security of an sAE scheme is defined with the following experiment between a challenger and an adversary $\mathcal{A}$:

1. The challenger samples $K \leftarrow \mathsf{StAE.Gen}()$, and $b \xleftarrow{\$} \{0,1\}$.
2. The adversary may adaptively query the encryption oracle Encrypt and the decryption oracle Decrypt, as described by Figure 2.1 (resp. Figure 2.2).
3. Finally, the adversary outputs its guess $b' \in \{0,1\}$ of b.

This game captures both the confidentiality and integrity properties of a sAE 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 2.5 (Secure sAE)**.** *The encryption scheme* StAE *is said to be a* secure stateful authenticated encryption scheme (sAE) *if, for all probabilistic polynomial time adversary* $\mathcal{A}$*,* $\mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}}(\mathcal{A})$ *is a negligible function in* k*.*

$\underline{\mathsf{Encrypt}(M, H)}$
$u \leftarrow u + 1$
$M_0 \xleftarrow{\$} \{0,1\}^{|M|}$
$M_1 \leftarrow M$
$(C^b, st_e^b) \xleftarrow{\$} \mathsf{StAE.Enc}(K, H, M_b, st_e)$
if $C^b = \perp$ then return $\perp$
$(C_u, H_u, st_e) \leftarrow (C^b, H, st_e^b)$
return C_u

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

Figure 2.1 – The Encrypt and Decrypt oracles in the sAE (RoR) security experiment. The counters u and v are initialised to 0, and $sync$ to `true` at the beginning of the experiment.

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

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

Figure 2.2 – The Encrypt and Decrypt oracles in the sAE (LoR) security experiment. The counters u and v are initialised to 0, and $sync$ to `true` at the beginning of the experiment.

2.3 Security Models

In this section, we present two classic security models. We start with the security model for authenticated key exchange protocols between two parties, that we call AKE model, as described

by Brzuska, Jacobsen, and Stebila [BJS16] (Section 2.3.1). We follow with the *Authenticated and Confidential Channel Establishment* (ACCE) security model proposed by Jager, Kohlar, Schäge, and Schwenk [JKSS11]. This model aims at analysing protocols when the security cannot be based on the indistinguishability of the session keys (Section 2.3.2).

The AKE model is one of the two main provable security tools that we employ in our work. We use the AKE model first to prove the security of our two-party key exchange protocol SAKE in Chapter 6, and as a building block in order to devise an extended security model (that we call 3-AKE) described in Chapter 5, Section 5.2. In Section 2.3.2 we present the second main tool that we employ: the ACCE model. The latter is used in Section 5.3 to prove the security of a protocol aiming at establishing secure tunnels, and also as a building block to devise an extension of the ACCE model intended to the three-party case (that we call 3-ACCE).

Besides the AKE and ACCE models that we mainly use, several other security models have been formerly proposed. We recall some of them in Section 5.1.

2.3.1 AKE Security Model

In this section, we present the security model for authenticated key exchange protocols described by Brzuska, Jacobsen, and Stebila [BJS16], that we call AKE model. This model incorporates all the features that are usually considered when analysing key agreement protocols in the public-key setting (e.g., DH-based protocols with signature). In this model, the adversary has full control over the communication network. It can forward, alter, drop any message exchanged by honest parties, or insert new messages. Brzuska et al.'s model then captures adaptive corruptions but also forward secrecy. This appears in the definition of the security experiment.

2.3.1.1 Execution Environment

Parties. A two-party protocol is carried out by a set of parties $\mathcal{P} = \{P_0, \ldots, P_{n-1}\}$. Each party P_i has an associated long-term key $P_i.\mathsf{ltk}$. The nature of the long-term key is not specified here. It can be a pair of public and private keys, a symmetric key, or a combination of both types.

Instances. Each party can take part in multiple (sequential or parallel) executions of the protocol. Each run of the protocol is called a session. To each session of a party P_i, an instance π_i^s is associated which embodies this (local) session's execution of the protocol, and has access to the long-term key of the party. In addition, each instance maintains the following state specific to the session.

- ρ: the role $\rho \in \{\mathsf{init}, \mathsf{resp}\}$ of the session in the protocol execution, being either the initiator or the responder.
- pid: the identity $\mathsf{pid} \in \mathcal{P}$ of the intended communication partner of π_i^s.
- α: the state $\alpha \in \{\perp, \texttt{running}, \texttt{accepted}, \texttt{rejected}\}$ of the instance.
- sk: the session key derived by π_i^s.
- κ: the status $\kappa \in \{\perp, \texttt{revealed}\}$ of the session key $\pi_i^s.\mathsf{sk}$.
- sid: the identifier of the session.
- b: a random bit $\mathsf{b} \in \{0, 1\}$ sampled at initialisation of π_i^s.

We put the following correctness requirements on the variables α, sk, sid and pid. For any two instances π_i^s, π_j^t, the following must hold:

$$(\pi_i^s.\alpha = \texttt{accepted}) \Rightarrow (\pi_i^s.\mathsf{sk} \neq \perp \wedge \pi_i^s.\mathsf{sid} \neq \perp) \tag{2.1}$$

$$\left(\pi_i^s.\alpha = \pi_j^t.\alpha = \texttt{accepted} \wedge \pi_i^s.\mathsf{sid} = \pi_j^t.\mathsf{sid}\right) \Rightarrow \begin{cases} \pi_i^s.\mathsf{sk} = \pi_j^t.\mathsf{sk} \\ \pi_i^s.\mathsf{pid} = P_j \\ \pi_j^t.\mathsf{pid} = P_i \end{cases} \tag{2.2}$$

Adversarial queries. The adversary $\mathcal{A}$ is assumed to control the network, and interacts with the instances by issuing the following queries to them.

- NewSession$(P_i, \rho, \mathsf{pid})$: this query creates a new instance π_i^s at party P_i, having role ρ, and intended partner pid. The instance's state is set to $\pi_i^s.\alpha = \texttt{running}$ and, if $\rho = \mathsf{init}$, the first message of the protocol is produced and returned to the adversary.
- Send(π_i^s, m): this query allows the adversary to send any message m to π_i^s. If $\pi_i^s.\alpha \neq \texttt{running}$, it returns $\perp$. Otherwise π_i^s responds according to the protocol specification.
- Corrupt(P_i): this query returns the long-term key $P_i.\mathsf{ltk}$ of P_i. If Corrupt(P_i) is the ν-th query issued by the adversary, then we say that P_i is *ν-corrupted*. For a party that is not corrupted, we define $\nu = +\infty$.
- Reveal(π_i^s): this query returns the session key $\pi_i^s.\mathsf{sk}$, and $\pi_i^s.\kappa$ is set to `revealed`.
- Test(π_i^s): this query may be asked only once throughout the game. If $\pi_i^s.\alpha \neq \texttt{accepted}$, then it returns $\perp$. Otherwise it samples an independent key $sk_0 \xleftarrow{\$} \mathcal{K}$, and returns sk_b, where $sk_1 = \pi_i^s.\mathsf{sk}$. The key sk_b is called the Test-*challenge*.

2.3.1.2 Security Definitions

Definition 2.6 (Partnership)**.** *Two instances π_i^s and π_j^t are* partners *if $\pi_i^s.\mathsf{sid} = \pi_j^t.\mathsf{sid}$.*

Definition 2.7 (Freshness)**.** *An instance π_i^s is said to be* fresh *with intended partner P_j, if*

(a) $\pi_i^s.\alpha = \texttt{accepted}$ and $\pi_i^s.\mathsf{pid} = P_j$ when $\mathcal{A}$ issues its ν_0-th query,

(b) $\pi_i^s.\kappa \neq \texttt{revealed}$ and P_i is ν-corrupted with $\nu_0 < \nu$, and

(c) for any partner instance π_j^t of π_i^s, we have that $\pi_j^t.\kappa \neq \texttt{revealed}$ and P_j is ν'-corrupted with $\nu_0 < \nu'$.

Note that the notion of freshness incorporates a requirement for forward secrecy.

An *authenticated key exchange protocol* (AKE) is a two-party protocol satisfying the correctness requirements (2.1) and (2.2), and where the security is defined in terms of an AKE experiment played between a challenger and an adversary. This experiment uses the execution environment described above. The adversary can win the AKE experiment in one of two ways: (i) by making an instance accept maliciously (Definition 2.8), or (ii) by guessing the secret bit of the Test-instance (Definition 2.9).

Definition 2.8 (Entity Authentication (EA))**.** *An instance π_i^s of a protocol Π is said to have* accepted maliciously *in the AKE security experiment with intended partner P_j, if*

(a) $\pi_i^s.\alpha = \texttt{accepted}$ *and* $\pi_i^s.\mathsf{pid} = P_j$ *when* $\mathcal{A}$ *issues its* ν_0*-th query,*

(b) P_i *and* P_j *are* ν*- and* ν'*-corrupted with* $\nu_0 < \nu, \nu'$*, and*

(c) there is no unique instance π_j^t *such that* π_i^s *and* π_j^t *are partners.*

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

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

Definition 2.9 (Key Indistinguishability)**.** *An adversary* $\mathcal{A}$ *against a protocol* Π*, that issues its* Test*-query to instance* π_i^s *during the AKE security experiment,* answers the Test-challenge correctly *if it terminates with output* b'*, such that*

(a) π_i^s *is fresh with some intended partner* P_j*, and*

(b) $\pi_i^s.\mathsf{b} = b'$*.*

The adversary's advantage is defined as

$$\mathsf{adv}_{\Pi}^{\mathsf{key\text{-}ind}}(\mathcal{A}) = \left| \Pr[\pi_i^s.\mathsf{b} = b'] - \frac{1}{2} \right|.$$

Definitions 2.8 and 2.9 allow the adversary to corrupt an instance involved in the security experiment (once the targeted instance has accepted, in order to exclude trivial attacks). Therefore, protocols secure with respect to Definition 2.10 below provide *forward secrecy*. Note that we do not allow the targeted instance to be corrupted before it accepts. That is, this security model does not capture key-compromise impersonation attacks (KCI) [BWJM97].[1]

Definition 2.10 (AKE Security)**.** *We say that a two-party protocol* Π *is a secure AKE protocol if* Π *satisfies the correctness requirements (2.1) and (2.2), and for all probabilistic polynomial time adversary* $\mathcal{A}$*,* $\mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A})$ *and* $\mathsf{adv}_{\Pi}^{\mathsf{key\text{-}ind}}(\mathcal{A})$ *are a negligible function of the security parameter.*

2.3.2 ACCE Security Model

In this section we present the *Authenticated and Confidential Channel Establishment* (ACCE) security model of Jager, Kohlar, Schäge, and Schwenk [JKSS11].

The AKE model described in Section 2.3.1 guarantees that the session key output by a key establishment protocol is indistinguishable from a random value. However there exist protocols where the session key is used during the key exchange phase. A famous example of such protocols is TLS 1.2 [DR08]. In TLS 1.2 the session key is used to protect messages (called Finished) which finalise the key exchange phase. Since these encrypted and MAC-ed Finished messages provide a trivial way to distinguish the session key from a random value, the security of TLS 1.2 cannot be proved based on indistinguishability of keys. The notion of ACCE is an alternative security model devised to circumvent the impossibility of using the AKE model in order to prove the security of TLS 1.2.

The first property captured by the ACCE model is entity authentication (as in the AKE model). But instead of guaranteeing that the session key is suitable to be used for any purpose, the second property ensures that the key is suitable for a specific purpose which is to set up an *authenticated and confidential channel*. More precisely, in the Jager et al.'s security model the second property corresponds to what the secure channel itself is supposed to achieve. That is,

[1] In this thesis, we consider protocols not resistant to KCI attacks.

authenticity and confidentiality of data. Hence an ACCE protocol allows establishing a secure tunnel in the sense of stateful authenticated encryption (see Section 2.2.5).

We present a slightly modified version of the original model of Jager et al. in order to adequately capture the security properties of the protocols we analyse in Chapter 3.

2.3.2.1 Execution Environment

The definitions of *parties* and *instances* are the same as in the AKE model (Section 2.3.1.1). We also consider the same correctness requirements (2.1) and (2.2). We define the following queries given to an adversary in order to interact with the instances and the parties during the security experiments.

Adversarial queries. The adversary $\mathcal{A}$ is assumed to control the network, and interacts with the instances by issuing the following queries to them.

- $\mathsf{NewSession}(P_i, \rho, \mathsf{pid})$, $\mathsf{Send}(\pi_i^s, m)$, $\mathsf{Corrupt}(P_i)$, $\mathsf{Reveal}(\pi_i^s)$: these queries are identical to the corresponding queries in the AKE model.
- $\mathsf{Encrypt}(\pi_i^s, M_0, M_1, H)$: this query encrypts the message M_b, $b = \pi_i^s.\mathsf{b}$, with header H, with the encryption session keys (stored within $\pi_i^s.\mathsf{ck}$) of an *accepting* instance π_i^s. If $\pi_i^s.\alpha \neq \texttt{accepted}$, then π_i^s returns $\perp$. Precisely, it proceeds as depicted by Figure 2.3, depending on the bit $\pi_i^s.\mathsf{b}$ sampled at random at the beginning of the security experiment.
- $\mathsf{Decrypt}(\pi_i^s, C, H)$: this query decrypts the ciphertext C with header H, with the decryption session keys (stored within $\pi_i^s.\mathsf{ck}$) of an *accepting* instance π_i^s. If $\pi_i^s.\alpha \neq \texttt{accepted}$, then π_i^s returns $\perp$. Precisely, it proceeds as depicted by Figure 2.3, depending on the bit $\pi_i^s.\mathsf{b}$ sampled at random at the beginning of the security experiment.

$\underline{\mathsf{Encrypt}(\pi_i^s, M_0, M_1, H)}$
if $\pi_i^s.\alpha \neq \texttt{accepted}$ 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 $\perp$
$b \leftarrow \pi_i^s.\mathsf{b}$
$(C_u, H_u, st_e) \leftarrow (C^b, H, st_e^b)$
return C_u

$\underline{\mathsf{Decrypt}(\pi_i^s, C, H)}$
if $\pi_i^s.\alpha \neq \texttt{accepted}$ then return $\perp$
if $\pi_i^s.\mathsf{b} = 0$ 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 $sync \leftarrow \texttt{false}$
if $sync = \texttt{false}$ then return M
return $\perp$

Figure 2.3 – The Encrypt and Decrypt oracles in the ACCE security experiment. StAE is the stateful authenticated encryption scheme used to establish the secure tunnel. The counters u and v are initialised to 0, and $sync$ to `true` at the beginning of every session. In case π_i^s does not have a partner when answering a Decrypt query, then $sync = \texttt{false}$.

We slightly relax the original model of Jager et al. as we do not require the StAE encryption scheme to hide the length of the plaintext.[2]

[2]The protocol we analyse in Chapter 3 with this model is built on a non length-hiding encryption function.

2.3.2.2 Security Definitions

An *authenticated and confidential channel establishment protocol* (ACCE) is a two-party protocol satisfying the correctness requirements (2.1) and (2.2) (see the AKE model in Section 2.3.1), and where the security is defined in terms of an ACCE experiment played between a challenger and an adversary. This experiment uses the execution environment described above. The adversary can win the ACCE experiment in one of two ways: (i) by making an instance accept maliciously (Definition 2.11), or (ii) by guessing the secret bit during the channel security experiment (Definition 2.12).

Definition 2.11 (Entity Authentication (EA))**.** *An instance π_i^s of a protocol Π is said to have* accepted maliciously *in the ACCE security experiment with intended partner P_j, if*

(a) $\pi_i^s.\alpha = \texttt{accepted}$ *and* $\pi_i^s.\mathsf{pid} = P_j$ *when $\mathcal{A}$ issues its ν_0-th query,*

(b) P_i *and* P_j *are ν- and ν'-corrupted with $\nu_0 < \nu, \nu'$,*

(c) $\pi'.\kappa \neq \texttt{revealed}$ *for any instance π' that accepted while having a matching conversation to π_i^s, and*

(d) there is no unique oracle π_j^t such that π_i^s has a matching conversation to π_j^t.

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

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

Definition 2.12 (Channel Security)**.** *An adversary $\mathcal{A}$ against a protocol* Π breaks the channel security *if it terminates the channel security game with a tuple (π_i^s, b') such that*

(a) $\pi_i^s.\alpha = \texttt{accepted}$ *and* $\pi_i^s.\mathsf{pid} = P_j$ *when $\mathcal{A}$ issues its ν_0-th query,*

(b) P_i *and* P_j *are ν- and ν'-corrupted with $\nu_0 < \nu, \nu'$,*

(c) $\pi_i^s.\kappa \neq \texttt{revealed}$*,*

(d) $\pi'.\kappa \neq \texttt{revealed}$ *for any instance π' such that π_i^s has a matching conversation to π', and*

(e) $\pi_i^s.\mathsf{b} = b'$*.*

The adversary's advantage is defined as

$$\mathsf{adv}_{\Pi}^{\mathsf{chan\text{-}sec}}(\mathcal{A}) = \left| \Pr[\pi_i^s.\mathsf{b} = b'] - \frac{1}{2} \right|.$$

Compared to the original model of Jager et al. we slightly change the EA and CS security definitions in the following way. We allow the targeted party P_i to be corrupted but only upon acceptance of its instance π_i^s. That is, resistance to KCI attacks is not captured (we do so in view of proving the security of a symmetric-key protocol, hence not resistant to KCI attacks, in Chapter 3). Nonetheless, the model captures the forward secrecy property (if $\nu \neq +\infty$ and $\nu' \neq +\infty$). Doing so, we obtain a model similar as the one used by Li, Schäge, Yang, Kohlar and Schwenk [LSY+14] to prove the security of TLS 1.2 in PSK mode (i.e., a symmetric-key version of TLS 1.2).

Definition 2.13 (ACCE Security)**.** *We say that a two-party protocol Π is a secure ACCE protocol if Π satisfies the correctness requirements (2.1) and (2.2), and for all probabilistic polynomial time adversary $\mathcal{A}$, $\mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A})$ and $\mathsf{adv}_{\Pi}^{\mathsf{chan\text{-}sec}}(\mathcal{A})$ are a negligible function of the security parameter.*

3 Analysis of LoRaWAN

LoRaWAN is an IoT security protocol deployed worldwide in more than 100 countries. LoRaWAN aims at securing communications between a back-end network and connected objects. These objects are intended to provide various types of services from telemetry measurement, remote device activation, site surveillance and intrusion detection, geolocation of valuable assets, up to management of networks that deliver sensitive resources (e.g., water, energy).

Two main versions of the protocol exist: 1.0 and 1.1. In this chapter, we provide an extensive analysis of 1.0, which is the currently deployed version, and we show that it suffers from several weaknesses. We introduce several attacks, including likely practical ones, that breach the network availability, data integrity, and data confidentiality. Finally we propose practical recommendations aiming at thwarting the attacks, while at the same time being compliant with the specification, and keeping the interoperability between patched and unmodified equipment.

Version 1.1 aims at correcting several flaws that impair the security of version 1.0. We present several weaknesses of LoRaWAN 1.1 that affect this latest release of the protocol. Then we present recommendations aiming at mitigating these flaws.

The results of this chapter have been published in [AF18b] and [CF19].

Contents

3.1 Introduction

3.1.1 Context

The communication protocol LoRa, developed by Semtech company, aims at setting up a Low-Power Wide-Area Network (LPWAN) based on a long-range, low-rate, wireless technology. It is somewhat similar to a cellular technology (2G/3G/4G mobile systems) but optimised for IoT/M2M. LoRa does not require a spectrum license because it uses free (although regulated) frequency bands (e.g., 863-870 MHz in Europe, 902-928 MHz in the USA, 779-787 MHz in China) [LoR17; Wor]. A LoRa end-device, with an autonomous power-supply, is supposed to communicate through several kilometers in an urban area, and to have a lifespan up to eight or ten years.

LoRaWAN is a protocol that aims at securing the Medium Access Control layer of a LoRa network. It is designed by the LoRa Alliance, which is an association that gathers more than 500 members (telecom operators, semiconductor manufacturers, digital security companies, hardware manufacturers, network suppliers, etc.) [IoT19].

Public and private LoRaWAN networks are deployed in more than 100 countries worldwide [IoT19] by telecom operators (SK Telecom, FastNet, ZTE, KPN, Orange, Proximus, etc.), private providers (e.g., LORIOT.io[1]), and private initiatives (e.g., The Things Network[2]). Several nation-wide networks are already deployed in Europe (France, Netherlands) [Fea16], Asia (South Korea) [Mar16], Africa (South Africa) [Biz15], Oceania (New Zealand) [Sma16b], providing already coverage to half of the population. Trials are launched in Japan [Bri16], the USA (starting with a hundred cities) [Kin16], China (the expected coverage extend to 100 million homes and 300 million people) [Sma16a], India (the first phase network aims at covering 400 million people across the country) [Kim16]. Figure 3.1 represents a map of LoRa networks deployed worldwide (in May 2017).

The use cases [OA16] that LoRaWAN aims at responding to include smart metering (electricity, water), tracking (shipping containers, valuable assets), agriculture (irrigation), smart grids (fault management), industrial (earthquake sensors, avalanche, flooding), smart city, wearables and health (medical wearables), connected home (security systems), vehicle telematics (traffic information, vehicle status).

In this chapter we focus on the two main versions of LoRaWAN that are defined:

- version 1.0.2 [SLE+16] released in 2016, which is the version currently deployed worldwide, and whose official name is now 1.0 (Sections 3.2-3.4),
- version 1.1 [Sor17] which aims at being the successor of version 1.0 (Sections 3.5-3.6).

In addition, version 1.0.3 [LoR18a] has been published in 2018, but it is cryptographically equal to version 1.0.2.

3.1.2 Attacks and Vulnerabilities

We show in Section 3.3 how to perform several attacks against LoRaWAN 1.0, including likely practical ones. These attacks allow to breach the network availability, data integrity, and data confidentiality, and target either an end-device or the back-end network. Based on the inner weaknesses of the protocol, these attacks do not lean on potential software or hardware bugs. Likewise they do not entail a physical access to the targeted equipment and are independent

[1] https://www.loriot.io
[2] https://www.thethingsnetwork.org

Figure 3.1 – Worlwide map of LoRa networks in May 2017 (source: `http://iot.semtech.com`)

from the means used to physically protect secret parameters. Several recommendations aiming at thwarting these attacks are presented in Section 3.4. In Section 3.6, we show that the latest version 1.1 of LoRaWAN still suffers from several flaws. We describe in Section 3.7 several ways to mitigate these vulnerabilities.

3.2 Protocol LoRaWAN 1.0

3.2.1 Overview

A LoRaWAN network corresponds to a star-of-stars topology: a set of end-devices (ED) communicates with several gateways which relay the data to a Network Server (NS) in the back-end side. In turn NS delivers the data to one or more Application Servers (AS) which own the corresponding ED, optionally through intermediary servers such as an MQTT server (see Figure 3.2). The security mechanisms are based on a symmetric key (the master key) MK_1 shared between an ED and NS. From this key, distinct per ED, two session keys are computed: the application session key K_a^e guarantees the data confidentiality between ED and AS; the command session key K_c^e guarantees the data integrity between ED and NS (thus data integrity is not end-to-end provided between ED and AS[3]). When a frame is exchanged exclusively between ED and NS, both data confidentiality and data integrity are provided by the command session key K_c^e. An application payload, if present, is always encrypted. If no payload is carried the frame is only authenticated. Encryption is done with AES [Nat01] in CTR mode [DH79; Dwo01], and data integrity is provided with a tweaked version of AES in CMAC mode [Dwo05; SPLI06] (a prefix block is added to the input). ED may establish an "activation" (namely a session) with NS through two ways. The pre-personalization ("Activation by Personalization", ABP) consists in setting two session keys (and other parameters but not the MK_1 master key) into ED before its deployment. An ED in ABP mode is then able to communicate with NS (and its AS) but not to renew the "session" keys. The other possibility ("Over the Air Activation", OTAA) consists

[3] As acknowledged by the specification ([SLE+16], §6.1.4).

in provisioning ED with an MK_1 master key and other parameters, allowing to perform key exchanges with NS through the radio interface once it is deployed. In this chapter, we focus on OTAA mode.

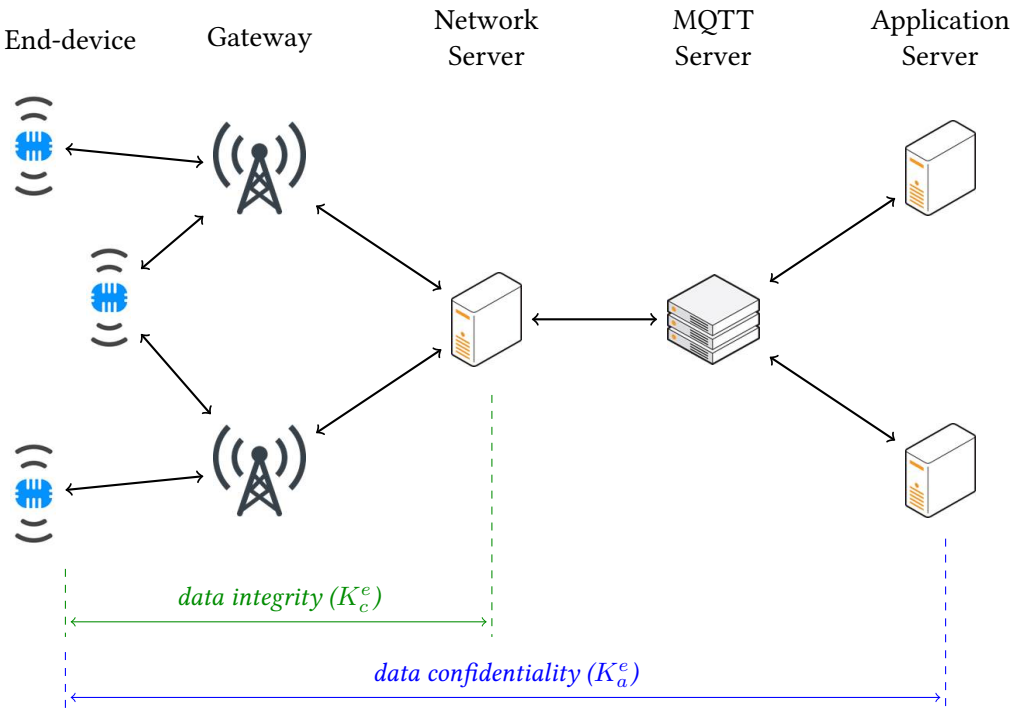


Figure 3.2 – LoRaWAN network in version 1.0. Data exchanged between ED and NS are encrypted with K_c^e. Data exchanged between ED and AS are encrypted with K_a^e.

3.2.2 Key Exchange

The key exchange done over the air is triggered when ED sends a *Join Request* message which NS responds to with a *Join Accept* message (see Figure 3.3). The (unencrypted) Join Request message includes two static IEEE EUI-64 identifiers (the ED's id_E, and the AS' id_A), and a pseudo-random value rnd_E generated by ED. The message is protected with a 32-bit CMAC authentication tag computed with the 128-bit (static) master key MK_1. The Join Accept response from NS contains the (static) identifier of the latter (id_N), a pseudo-random value generated by NS (rnd_N), a value used as ED short address ($DevAddr$), and several (optional) radio parameters. The Join Accept message is protected with a CMAC authentication tag, and encrypted with AES (both operations made with the master key MK_1).[4] Two 128-bit session keys are then computed:

$$\begin{aligned} K_c^e &= \mathsf{AES}(MK_1, \texttt{0x01} \| data) \\ K_a^e &= \mathsf{AES}(MK_1, \texttt{0x02} \| data) \end{aligned}$$

with

$$data = rnd_N\ (3) \| id_N\ (3) \| rnd_E\ (2) \| \texttt{0x00} \cdots \texttt{00}\ (7).$$

Thus the session keys depend mostly on a secret and static value (the master key MK_1), and two pseudo-random values of 16 and 24 bits. Once the Join Request and Join Accept

[4] More precisely the AES decryption function is used to protect the Join Accept message, since ED implements the encryption function only.

messages are exchanged, ED, NS and AS are able to communicate. After NS computes the session keys, it transmits the application session key K_a^e to AS, which has thus no control on this key sharing phase, entirely handled by NS. The NS must keep the previous session keys, and the corresponding security parameters, until it receives a (valid) frame protected by the new security parameters. The security mechanisms between NS and AS are out of the LoRaWAN scope.

ED		NS
(MK_1)		(MK_1)
$rnd_E \xleftarrow{\$} \{0,1\}^{16}$		
$\tau_E \leftarrow \mathsf{MAC}(MK_1, id_A \| id_E \| rnd_E)$		
Join Request $\leftarrow id_A \| id_E \| rnd_E \| \tau_E$		
	Join Request $\longrightarrow$	
		Verify τ_E
		$rnd_N \xleftarrow{\$} \{0,1\}^{24}$
		$\tau_N \leftarrow \mathsf{MAC}(MK_1, rnd_N \| id_N \| DevAddr \| prms)$
		Join Accept $\leftarrow \mathsf{AES}^{-1}(MK_1, rnd_N \| id_N \| DevAddr \| prms \| \tau_N)$
		$data \leftarrow rnd_N \| id_N \| rnd_E \| \mathtt{00} \cdots \mathtt{00}$
		$K_c^e \leftarrow \mathsf{AES}(MK_1, \mathtt{0x01} \| data)$
		$K_a^e \leftarrow \mathsf{AES}(MK_1, \mathtt{0x02} \| data)$
	$\longleftarrow$ Join Accept	
$rnd_N \| id_N \| DevAddr \| prms \| \tau_N \leftarrow \mathsf{AES}(MK_1, \text{Join Accept})$		
Verify τ_N		
$data \leftarrow rnd_N \| id_N \| rnd_E \| \mathtt{00} \cdots \mathtt{00}$		
$K_c^e \leftarrow \mathsf{AES}(MK_1, \mathtt{0x01} \| data)$		
$K_a^e \leftarrow \mathsf{AES}(MK_1, \mathtt{0x02} \| data)$		
	post-accept phase $\longleftrightarrow$	

Figure 3.3 – Correct execution of LoRaWAN 1.0

3.2.3 Data Encryption and Authentication

In this section we describe the computations done in order to encrypt and provide integrity protection to a frame.

The plaintext frame payload $ptext$ is encrypted in CTR mode. From the following 16-byte block

$$A_i = \mathtt{0x01}\ (1) \| \mathtt{0x00} \cdots \mathtt{00}\ (4) \| dir\ (1) \| DevAddr\ (4) \| fcnt\ (4) \| \mathtt{0x00}\ (1) \| i\ (1)$$

a secret keystream $S_i = \mathsf{AES}(K, A_i)$, with $K \in \{K_a^e, K_c^e\}$, is produced and used to mask the payload:

$$ctext = (S_0 \| \cdots \| S_{n-1}) \oplus ptext.$$

The session key $K = K_a^e$ is used when *application* messages are exchanged between ED and AS, and $K = K_c^e$ is used when *command-only* messages are exchanged between ED and NS. dir specifies the direction (uplink = 0x00, downlink = 0x01). $fcnt$ is the frame counter (16 or 32 bits), initialised to 0 when the session starts, and monotonically increased when a (valid) frame is sent or received. Two different counters are used depending on the frame's direction. $DevAddr$ is ED's address (within a given LoRa network) chosen by NS and sent in the Join Accept message, and it remains constant during the entire session. To compute $DevAddr$, seven bits are chosen from the NS' unique identifier id_N: $\mathsf{msb}_7(DevAddr) = \mathsf{lsb}_7(id_N)$, and 25 bits are "*arbitrarily*" assigned by NS. The i value numbers the AES blocks within the payload to encrypt.

A 32-bit authentication tag τ is computed with CMAC and the command session key K_c^e on the whole frame (header hdr of size $hlen \in \{8, \ldots, 24\}$ and encrypted payload $ctext$ of size $plen$) and a 16-byte prefix block

$$B_0 = \texttt{0x49}\ (1) \| \texttt{0x00} \cdots \texttt{00}\ (4) \| dir\ (1) \| DevAddr\ (4) \| fcnt\ (4) \| \texttt{0x00}\ (1) \| (hlen + plen)\ (1).$$

Note that B_0 and A_i differ only on the first and last bytes, and share the same parameters $DevAddr$ and $fcnt$. The frame eventually sent is

$$hdr\ (hlen) \| ctext\ (plen) \| \tau\ (4).$$

Figure 3.4 depicts the generation of an application frame. The frame header hdr includes, among other fields, $DevAddr$, the frame counter $fcnt$ on 16 bits, and an (optional) field $FOpts$ which may contain commands exclusively exchanged between ED and NS. If the frame counter is 32-bit long, this $fcnt$ field corresponds to the least 16 significant bits. The commands included in the $FOpts$ field are in clear. If they have to be encrypted they must be included in the frame payload. In such a case the payload cannot contain application data, and the encryption key used is the command session key K_c^e.

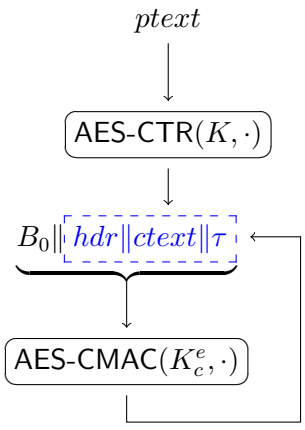


Figure 3.4 – Generation of an application frame in LoRaWAN 1.0. The session key $K \in \{K_a^e, K_c^e\}$ is used to encrypt the $ptext$ payload. The encrypted frame appears in the blue dashed box.

3.3 Attacks against LoRaWAN 1.0

Hereinafter we present the attacks we have found against the LoRaWAN protocol version 1.0. In Sections 3.3.1 and 3.3.2, the attacker, standing between ED and NS, needs only to act on the

air interface: she needs to eavesdrop on data exchanged between ED and the server, and to send data to any equipment. With these simple requirements, the attacker can replay frames, decrypt frames, and desynchronise an ED.

In Section 3.3.3, we describe other kind of attacks where the attacker needs to act at some point on the link between NS and AS. Then the attacker can replay, forge, and, possibly, decrypt frames.

3.3.1 Replay or Decrypt

In LoRaWAN, encryption is done in CTR mode [DH79; Dwo01] which security is proved by Bellare, Desai, Jokipii, and Rogaway [BDJR97]. Likewise CMAC mode (also known as OMAC1 [IK03a; IK03b]), used to compute a frame's authentication tag, is proved secure by Iwata and Kurosawa [IK03c], and Nandi [Nan09]. Of course this does not necessarily imply that a protocol based on these cryptographic primitives is secure in turn [Bel98; DPW11]. In particular the security of these encryption and authentication modes is no longer guaranteed in case of a misuse, namely if same session keys, counter blocks, and B_0 prefix block are reused. Based on the peculiarities of LoRaWAN, it is actually possible to compel ED or NS to reuse previous security parameters. We describe precisely how to perform such an attack against ED or NS, and its consequences.

3.3.1.1 Targeting the End-device (Attack A1)

Goal. The purpose of this attack is to compel ED to reuse previous session keys and other security parameters. When this happens, frames picked from a previous session become cryptographically valid anew, hence can be replayed. Moreover the same secret keystream is then used to protect the frames exchanged during the new session. This allows an adversary to decrypt frames.

Since ED ends up reusing previous session keys (which are no longer shared with NS), this attack is also a kind of "desynchronisation" attack. However, contrary to the desynchronisation attacks described in Section 3.3.2, this "replay or decrypt" attack has more devastating consequences (and a higher complexity) than merely desynchronising ED and NS.

Key points. The encryption keystream $S_i = \mathsf{AES}(K, A_i)$ used to protect a frame payload is produced from a session key $K \in \{K_a^e, K_c^e\}$ and A_i block counters. Within a given session the blocks

$$A_i = \texttt{0x01}\ (1) \| \texttt{0x00} \cdots \texttt{00}\ (4) \| dir\ (1) \| DevAddr\ (4) \| fcnt\ (4) \| \texttt{0x00}\ (1) \| i\ (1)$$

(as well as the prefix block B_0) depend mostly on the frame counter $fcnt$ (set to 0 when the session starts and monotonically increased frame after frame), and on the $DevAddr$ parameter (static during the whole session). The other parameters are the direction dir unchanged for a given direction, and the i block index which evolves the same way for each frame. Hence the way the keystream S_i changes depends only on the $DevAddr$ parameter and the session key (usually K_a^e). For a given ED, which connects to the same NS (hence uses the same static id_N parameter), the session keys depend mainly on a secret and static value (MK_1) and two pseudo-random values (rnd_E, rnd_N). Therefore, if one succeeds in compelling ED to reuse the same $DevAddr$, rnd_E and rnd_N parameters, this leads not only to the reuse of previous session keys K_a^e, and K_c^e, but also to the reuse of previous keystream S_i and prefix block B_0.

Attack. The purpose is to make ED use twice the same rnd_E, rnd_N, and $DevAddr$ values. The 16-bit rnd_E and 24-bit rnd_N parameters are pseudo-random. Let us assume that an attacker is able to impose the rnd_N value that ED uses to compute the session keys. Then the probability that the session keys repeat depends only on the rnd_E parameter. Firstly note that a collision due to the birthday paradox happens with high probability ($p = \frac{1}{2}$) after roughly $\sqrt{2\ln(2) \times 2^{16}} \simeq 301$ sessions only.

The attacker can speed up the whole process: she eavesdrops on a given session, and compels ED to generate multiple rnd_E values until the expected value is produced once again. In such a case only one value among 2^{16} is useful to the attacker. Hence, ED must generate on average 2^{16} rnd_E values. More generally, the attacker can eavesdrop on several different Join Accept messages sent by NS to ED, where each one corresponds to a different rnd_E value. Let n_{ja} be the number of Join Accept messages collected by the attacker (necessarily $n_{ja} \leq 2^{16}$ since there is at most 2^{16} different rnd_E values). Then the attacker compels ED to send a fresh Join Request message, and expects that it matches with one of the n_{ja} Join Accept messages previously collected. The probability of success is $p = n_{ja}/2^{16}$. The number of times that the attacker must repeat this experiment (i.e., compelling ED to send a new Join Request message) in order to be successful is then $1/p = 2^{16}/n_{ja}$. This means that ED has to send $2^{16}/n_{ja}$ Join Request messages before one carries a rnd_E value that matches with one of the Join Accept messages.

Once this first phase of the attack is achieved, the attacker ends with two different sessions protected with the same security parameters, denoted respectively s_{old} and s_{new} (see Figure 3.5).

Technique 1 used to achieve the attack: replay of a Join Accept message. In order to compel ED to use a given rnd_N value, the attacker can replay a previous Join Accept message sent to the targeted ED. Then ED will reuse (once again) the parameters included in the message. Indeed the data carried in a Join Accept message correspond to

$$rnd_N\ (3) \| id_N\ (3) \| DevAddr\ (4) \| prms\ (2\text{-}18) \| \tau_N\ (4)$$

where τ_N is an authentication tag computed on the preceding fields with the (static) master key MK_1. These parameters are protected with AES and MK_1. The cornerstone of this attack is that all the parameters are chosen by NS, in particular rnd_N and $DevAddr$. id_N is the NS' (static) identifier, and the $prms$ are also defined by NS. The only secret parameter involved in the message calculation is static (MK_1). Hence, ED is not able to verify if the received Join Accept message corresponds to the Join Request it sent. Replaying a Join Accept message allows the attacker to compel ED to (re)use both rnd_N and $DevAddr$ parameters.

The possible choices for the attacker bear on the Join Accept messages previsouly sent by NS to the targeted ED. A Join Accept message intended to another ED is not usable since the message is protected with ED's master key.

Technique 2 used to achieve the attack: harvest of Join messages. In order for the attacker to be successful, she must compel ED to generate multiple rnd_E values. This can be achieved through at least two attack vectors: first the air interface, second the power supply.

Attack vector: air interface. In this scenario, the ability of the attacker to make ED generate multiple rnd_E values is related to the behaviour of ED when it sends a Join Request message but does not receive a Join Accept response or receives an invalid message. The specification states that NS shall ignore Join Request messages containing previously used rnd_E values in order to thwart a replay attack ([SLE+16], §6.2.4). Hence, ED has to generate a new pseudo-random rnd_E value each time it computes a Join Request message, even when a previous Join Request message

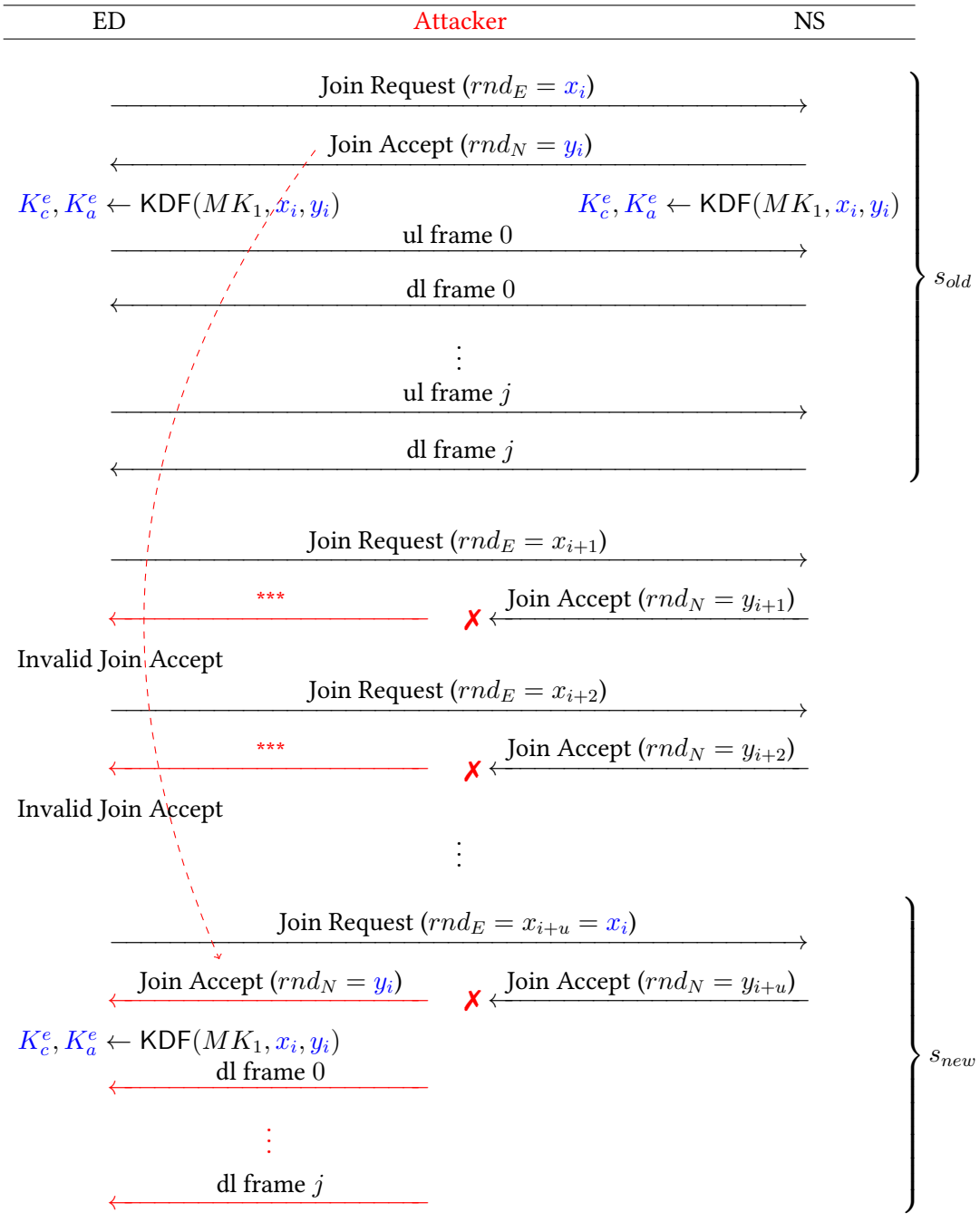


Figure 3.5 – "Replay or decrypt" attack against ED in the "air interface" scenario

did not receive a response. Otherwise ED may fear the subsequent Join Request messages to be dropped by NS. This allows the attacker to collect multiple new and valid Join Request messages. It is enough for the attacker to send invalid Join Accept messages in response to the ED's messages.

The shortest receiving window of a Join Accept message is 5 seconds [Wor]. If the attacker uses $n_{ja} = 16$ Join Accept messages, the attack is achieved after roughly $2^{16}/16 \times 5$ seconds $=$ 5.7 hours (assuming that the time needed to process the Join messages is negligible compared to the communication duration).

Attack vector: power supply. Alternatively, the attacker can influence on the power supply in order to compel ED to repeatedly reboot. Each time ED switches on, it sends a new Join Request message to connect the back-end network. If ED is self-powered, the attacker must be able to access the battery. If ED is powered by a continuous supply, the attacker can turn off or interrupt a remote electric generator ED is connected to, or the link between the generator and ED. At least two other means can also be considered. Firstly, electromagnetic impulses targeting ED can lead to a temporary power outage, and then a reboot. Secondly, the attacker can use a softest version of a "USB Killer" [Ant17]. This tool appears to be a regular USB key, but, once plugged into a device, it sends an electric impulse that destroys the device. If ED presents an available communication port (USB or, possibly, another type of port), a variant of this tool can be used to transmit low-intensity electric shocks to ED such that the latter reboots. In addition, this reboot tool could be remotely controlled.

It is conceivable that this second scenario be more efficient because the time elapsed before a new Join Request message be sent by ED depends only on the reboot duration.

This harvesting technique yields another advantage. If the attacker forbids NS from receiving the Join Request messages sent by ED, she gets fresh messages (i.e., unknown to NS) for free. Note that every time NS receives a Join Request message, it sends a new Join Accept message. Therefore, this procedure is also a way to collect multiples Join Accept messages.

Impact: frame replay. Frames drawn from the previous session (s_{old}) can be replayed to ED throughout the new session (s_{new}).[5] These frames are valid since they are protected with a cryptographically correct keystream and authentication tag. The attacker has to take care about the sequentiality. Indeed a frame shall be rejected by ED if its counter does not belong to $\{fcnt, \ldots, \texttt{MAX\_FCNT\_GAP}\}$, where $fcnt$ is the dowlink frame counter (in that specific case) managed locally by ED and used as reference (its initial value is 0), and $\texttt{MAX\_FCNT\_GAP} = 2^{14}$. Hence the attacker may virtually choose up to $2^{14} + 1$ frames in order to deceive ED (more precisely the number of available frames depends on the number of frames actually sent by NS or AS throughout session s_{old}). Note that the first frame the attacker replays may be any of these. However the subsequent replayed frames must have increasing counter values.

Impact: frame decryption. The frame payload is encrypted in CTR mode. Once the attack is achieved, ED uses twice the same keystream in order to protect different frames. The frame of counter t sent during session s_{old} contains an encrypted payload $c_t^{s_{old}} = m \oplus k_t^{s_{old}}$, where m is the plaintext and $k_t^{s_{old}}$ the keystream. The frame of same counter t sent during session s_{new} contains an encrypted payload $c_t^{s_{new}} = m' \oplus k_t^{s_{new}}$. Since $k_t^{s_{old}} = k_t^{s_{new}}$, we have that $c_t^{s_{old}} \oplus c_t^{s_{new}} = (m \oplus k_t^{s_{old}}) \oplus (m' \oplus k_t^{s_{new}}) = m \oplus m'$. Hence m and m' may (partially or

[5] We use the term "session", yet it is a misuse of language. Indeed this word does not depict precisely what are the actual exchanges since ED, at this point, has no "partner": neither NS nor AS is able to communicate with ED.

completely) be retrieved (in an obvious manner if one message, m or m', is known, or through analysis of $m \oplus m'$ [MWES06]).

According to the LoRaWAN specification ([SLE+16], §4.3.1.1), if ED sends to NS more than `ADR_ACK_LIMIT` $= 64$ frames without receiving any response, it has to ask an explicit acknowledgement to the server (ED sets the `ADRACKReq` bit to 1 within the frame header). The ED can then send up to `ADR_ACK_DELAY` $= 32$ more frames to the server. If still no frame has been received from the server, ED *may* switch to the next lower data rate that provides a longer radio range. Furthermore, if ED already uses its lowest available data rate, it *shall* not ask for such an acknowledgement. The specification provides no guidance on how ED shall behave in the latter case, or if it still does not receive an acknowledgement after it changed its data rate, or if it decides not to change its rate. We may reasonably assume that ED keeps sending frames. This means that the attacker has at her disposal at least `ADR_ACK_LIMIT` + `ADR_ACK_DELAY` $= 64+32 = 96$ frames usable for her decryption attempts. Moreover, if ED asks for an acknowledgement (the attacker is aware of that since the information `ADRACKReq` lies in the unencrypted frame header), the attacker can use any downlink frame drawn from session s_{old}, and replay it to ED. Indeed, according to the specification, this is enough to respond to the acknowledgement request sent by ED.

Comment on some mitigation mechanisms. The LoRaWAN specification mentions two mechanisms that can be seen at first sight as efficient ways to mitigate the "replay or decrypt" attack. Below we explain why these mechanisms are in fact not sufficient to thwart our attack.

Duty cycle. The ducty cyle is a mechanism used to regulate the occupation rate of the radio channel by ED. Enforcing the duty cycle implies that ED must wait some time before sending a new frame, hence cannot repeatedly send a lot of messages. Therefore one could claim that the duration of the attack is greater than the figure we provide. However, the duty cycle is a regulation mechanism, not a security one (even if it could cleverly be used as such). Secondly, not all countries compel to use such a mechanism (e.g., the USA and India do not). For instance, in the USA, there is no "global" duty cycle related to the 902-928 MHz frequency band, but ED must respect a dwell time that does not forbid from sending as many frames as wished. Finally, ED may well be certified (by the LoRa Alliance [LoR]) and yet not apply the duty cycle. Indeed the LoRa Alliance certification procedure does *not* cover any regulatory testing [Hun17].

Retransmissions back-off . The specification describes also a mechanism aiming at limiting the number of messages ED can send whereas the back-end network remains silent. The critical situations considered by the specification are for instance an earthquake or a power outage on the back-end side. If ED sends a message that requires a response but does not receive any (i.e., its receiving window remains empty), then it may everlastingly send new requests. If all the ED that are affiliated to the problematic back-end network behave in this manner, this ends with the air interface being flooded and eventually jammed with all the messages. Yet, this limitation mechanism is applied by ED when the back-end network remains silent. In our attack scenario, when ED sends a Join Request message it does receive a message that is classified (through its header) as a response (i.e., a Join Accept message), even though this (invalid) response is sent by the attacker. One could argue that a cryptographically invalid message is necessarily sent by an attacker since an integrity check (based on a CRC) is done at the radio level. However, regarding the downlink messages (received by ED), the specification states that "*no payload integrity check is done at this level to keep messages as short as possible with minimum impact on any duty-cycle limitations of the ISM bands used*" ([SLE+16], §3.2). Finally,

if this limitation mechanism is applied until ED receives a *valid* response from the back-end network, this invalidates the air interface vector, but the "replay or decrypt" attack remains still possible through the power supply vector, and leads to the same consequences.

3.3.1.2 Targeting the Network Server (Attack A2)

Goal. The same kind of attack can be performed against NS, aiming at compelling the server to use the same security parameters throughout two different sessions. The goal is then to compel NS to use twice the same rnd_E, rnd_N and $DevAddr$ values.

Attack: method 1. The key exchange is triggered by the Join Request message. Hence an attacker replaying such a message sets the rnd_E parameter before knowing the $DevAddr$ and rnd_N values NS generates. These two last parameters must correspond to the rnd_E value chosen by the attacker, hence only one such pair among all possible values is of interest to the attacker.

According to the specification, NS must keep track of "*a certain number*" of received rnd_E values in order to prevent replay attacks, without clarifying if this means all values or a few of them. We may reasonably assume that NS keeps track of a few values (say n), which seems confirmed by several open source codes (e.g., [Ttn; Got]). Thus the attacker cannot choose any Join Request she wants to replay. The corresponding rnd_E value must not belong to the list of n stored values. If the value the attacker wants to replay still belongs to the server's list (let i be its index, with 0 and $n-1$ the index of the oldest and of the latest received values), she has to wait for $i+1$ additional (legitimate) key exchanges before NS "forget" that value. The duration of such an "opportunist" attack depends on the frequency of the key exchanges.

rnd_N is a 24-bit pseudo-random value. The 32-bit $DevAddr$ parameter is made of 7 bits from id_N, and 25 bits which are "*arbitrarily*" chosen by NS ([SLE+16], §6.1.1). If $DevAddr$ is pseudo-random then the probability of success is $2^{-(24+25)} = 2^{-49}$. But "*arbitrarily*" does not mean "pseudo-random", and observations of real-life sessions established between several ED and NS show that the $DevAddr$ parameter remains unchanged for a given ED throughout different sessions. For instance, some NS implementation derives the $DevAddr$ parameter from the unique ED's identifier id_E. Also the $DevAddr$ value may be chosen once and for all at the time of ED provisioning. In such a case the probability of success increases to 2^{-24}, and the overall probability of success is 2^{-24} every $n+1$ sessions. Alternatively, the attacker can eavesdrop on n_{jr} different Join Request messages (that NS has "forgotten"), and send them to the server. Note that the messages may come from different ED, hence, may have to be sent to one or several NS servers. The probability that at least one message triggers the same rnd_N value as during a previous session is $1-(1-2^{-24})^{n_{jr}} \simeq n_{jr} \times 2^{-24}$. For instance, if the attacker uses $n_{jr} = 2048$ Join Request messages, her probability to succeed raises to $\frac{1}{8192}$.

The attacker is successful when NS sends the expected rnd_N value. But, contrary to a Join Request message (sent in clear), a Join Accept message is protected with AES^{-1} in ECB mode. Before encryption a Join Accept message corresponds to

$$rnd_N\ (3) \| id_N\ (3) \| DevAddr\ (4) \| prms\ (2\text{-}18) \| \tau_N\ (4).$$

id_N is the unique NS' identifier, hence static. As said, the $DevAddr$ parameter assigned to a given ED remains unchanged. The $prms$ (frequency plan) depend on the gateway, hence likely remain the same for quite a long time. And τ_N is an authentication tag computed on the preceding values with the ED's (static) master key. Hence only rnd_N may vary from one Join Accept message to another. Through direct comparison between a Join Accept message

(received during the attack) and the one used as reference (beforehand eavesdropped by the attacker during session s_{old}), the attacker is able to check if the rnd_N value repeats.

Attack: method 2. Another method can speed up the attack. In a first phase the attacker collects a set of $n + k$, $k \geq 1$, different Join Request messages from a given ED. Necessarily at least one of these messages contains a rnd_E value which is "forgotten" by NS. The other messages may carry rnd_E values known at the moment to NS. These messages must be ordered in the following way: first the "forgotten" messages, followed by the others sorted in the same relative order as in the NS' list.[6] In a second phase the attacker sends continuously each Join Request message of its circular list. In response to each request NS responds with a Join Accept message. The attacker keeps starting new sessions until she receives twice the same rnd_N value (thanks to the birthday paradox). Then the attacker gets two different sessions protected with the same security parameters (if the $DevAddr$ parameter remains unchanged). In order to get twice the same rnd_N value with high probability ($p = \frac{1}{2}$), it is enough for NS to generate roughly $\sqrt{2\ln(2) \times 2^{24}} \simeq 4{,}823$ Join Accept messages per Join Request message. To achieve this the attacker must start $(n + k) \times \sqrt{2\ln(2) \times 2^{24}}$ sessions. For instance, if NS keeps track of $n = 10$ rnd_E values, the attacker can use a list of $n + 1 = 11$ Join Request messages, and the number of sessions needed to achieve the attack is $53{,}049$. This corresponds to less than 74 hours (at the rate of 5 seconds per key exchange). However to get twice the same security parameters without application frames protected with these parameters is pointless. Hence, during the second phase of this method the attacker has to let NS send several frames before starting a new session. Therefore the duration of the whole attack is likely greater than 74 hours.

Technique used to achieve the attack. In order to collect the Join Request messages, the attacker can iterate $n + k$ times the procedure described in Section 3.3.1.1. Then she gets a list of messages correctly sorted and ready to be used. The Join Request messages usable by the attacker must come from the same ED since the session keys are computed with the ED's master key (and also if the $DevAddr$ parameter is closely related to ED – e.g., computed from its id_E identifier).

Impact. Once the attacker succeeds in compelling NS to compute once again the same security parameters, she eventually gets two different sessions (s_{old} and s_{new}) protected with the same security parameters. The attacker is then able to replay uplink frames and attempt decryption of downlink frames.

According to the specification, when a new key exchange is done NS shall keep the previous security parameters until it receives a (valid) frame protected with the new security parameters, and then it can remove the previous ones ([SLE+16], §6.2.4). Yet, since the session keys and other security parameters are reused, the attacker can easily replay to NS a frame drawn from the previous session s_{old}, thus "confirming" the new keys to NS. Then the server drops the current keys and is ready to use the new ones.

[6] More exactly the necessary condition is the following: each message collected by the attacker and common with the NS' list must have a greater index than the index in the server's list (0 being the index of the oldest message, $n - 1$ the index of the latest), so that the message is replayed once it has left the server's list.

3.3.2 Desynchronisation

In this section, we present another kind of attack, based on as little as a single message replay in one of the two scenarios we describe, which aims at lastingly precluding ED and NS from communicating.

3.3.2.1 Targeting the End-device (Attack A3)

Goal. This attack aims at "disconnecting" ED from the network. That is ED performs a successful key exchange which ends with ED not sharing the new session keys with NS (ED has no "partner"). Therefore the frames sent by ED are ignored by NS, and conversely.

Key points. The session keys are computed, by a given ED and NS, with two static parameters (the NS' unique identifier id_N, and the ED's master key MK_1), and two variable parameters (the pseudo-random values rnd_N computed by NS, and rnd_E by ED). As soon as ED receives a (valid) Join Accept message it can derive the session keys and start transmitting protected frames. In the key derivation, if ED uses values different from those actually sent by NS (say $(rnd_E, rnd_N) = (x, \tilde{y})$ on the one hand, and $(rnd_E, rnd_N) = (x, y)$, on the other hand, $y \neq \tilde{y}$), it eventually computes different session keys than those computed by the server. This does not forbid ED from sending protected frames though. However those frames will be dropped by NS since they are invalid from the server perspective. Conversely, the frames sent by NS will be discarded by ED. Thus ED, unable to communicate with NS, is "disconnected" from the network.

Attack. In order to perform such a desynchronisation attack, an attacker can first passively eavesdrop on a Join Accept message sent by NS in response to ED's Join Request message. When ED starts a new session and sends another Join Request message, the attacker replies before NS and replays the eavesdropped Join Accept message. This replayed message likely contains an $rnd_N = \tilde{y}$ value different from the fresh one sent by NS ($rnd_N = y$). Hence, ED and NS compute different session keys and security parameters (see Figure 3.6).

Technique used to achieve the attack. The attacker is able to replay a previous Join Accept message thanks to the peculiarities of the LoRaWAN protocol: indeed ED has no means to verify neither if the message is a replay, nor if it is an actual response to the Join Request message it just sent. Moreover the attacker can use the procedure described in Section 3.3.1.1 to collect several Join Accept messages and use these "desynchronisation ammunition" anytime later. The Join Accept message used by the attacker must be intended to the targeted ED. Indeed such a message is protected with the master key of the ED it is sent to.

Impact. Such a desynchronisation attack may be harmful because it can lastingly disturb the operating of a LoRaWAN network. The usual behaviour of, say, a sensor may be to regularly send some measurements without expecting a response unless the server detects an anomaly in the collected data. If ED sends its measurement at a low rate, days or even weeks may elapse before something abnormal is noticed, even if ED is supposed to react if it does not receive a downlink frame after a fixed number of sent frames. For instance, if ED sends one frame per hour, at least four days (`ADR_ACK_LIMIT` + `ADR_ACK_DELAY` $= 64 + 32 = 96$ hours) may elapse.

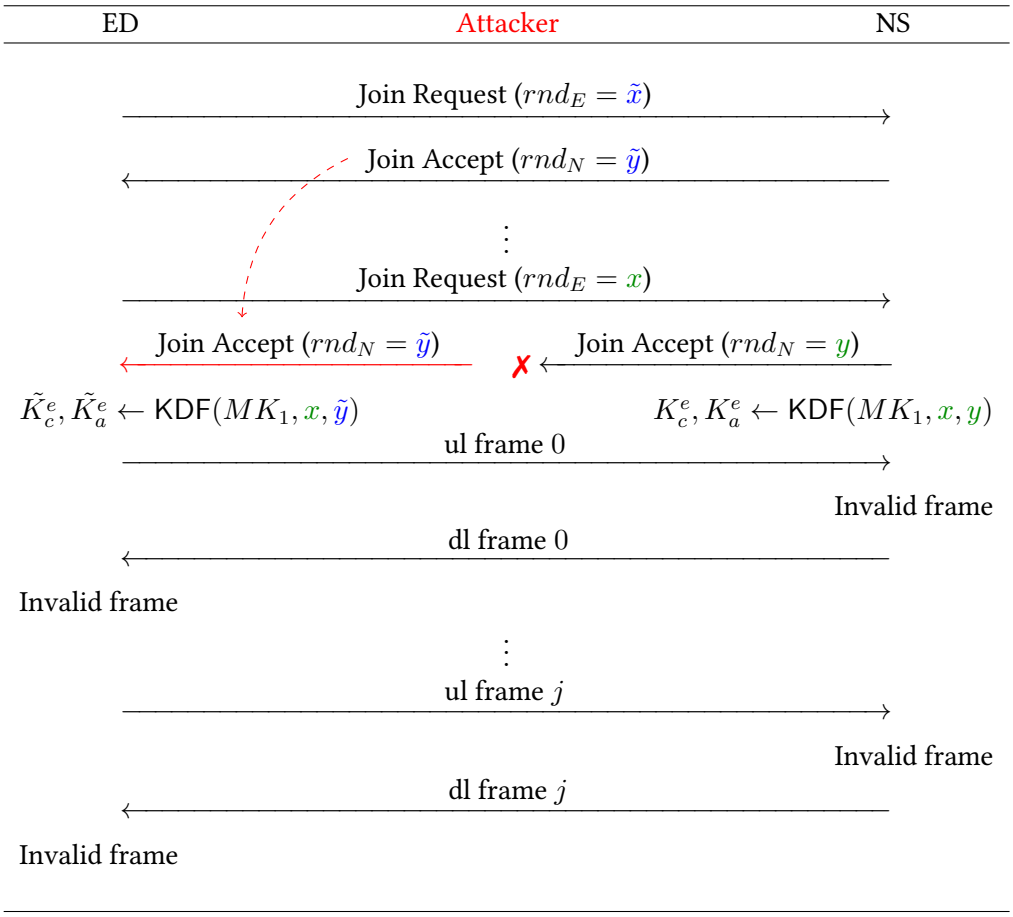


Figure 3.6 – Desynchronisation attack against ED

3.3.2.2 Targeting the Network Server (Attack A4)

Goal. The same kind of desynchronisation attack can be done against NS, aiming at disconnecting a given ED from the network. In that case, NS completes the key exchange without being "partnered" with the intended ED (i.e., identified by the id_E parameter within the Join Request message). Therefore the frames NS (and AS) may send are ignored by ED, and conversely.

Attack. Upon reception of a (valid) Join Request message, NS generates a new rnd_N value and computes new session keys. If an attacker succeeds in replaying to NS a valid Join Request message, the corresponding ED will no longer share the same session keys with NS.

If the attacker replays a previous Join Request message, it may be rejected since NS is supposed to keep track of previously received rnd_E values. This means that the attacker has to expect, or to wait, for the rnd_E value included in the replayed Join Request message to no longer belong to the server's list (i.e., the attacker has to wait for the targeted ED to start enough sessions so that the server "forgets" that rnd_E value). Alternatively the attacker may send a brand new Join Request message to be sure that it is not rejected by NS. However, the message is protected by a 32-bit authentication tag computed with the ED's master key MK_1. Hence the probability for the attacker to forge such a valid message is 2^{-32}.

A trade-off is the following: the attacker uses a fresh Join Request message. She leans on the targeted ED to compute such a message, while forbidding NS from receiving it (at the moment of its collection). Therefore the message is at the same time valid and unknown to NS.

The attacker has another challenge to take up. The specification states that NS must keep the previous session keys (and the corresponding counters) until it receives a valid frame protected with the new keys, and then it can drop the previous security parameters and keep only the new ones.[7] Yet it is unclear about how NS should behave if it receives successively several valid Join Request messages but no frames protected with any of the new computed session keys. We may assume that NS stores at most a few number of security parameters. Let us assume that NS stores only two sets of session keys: the latest valid one, and the latest computed one. Let $seskey_i$ be the current (valid) session keys (used by ED and NS to exchange frames). The attacker can do the following. She waits for ED to start a new session. New session keys ($seskeys_{i+1}$) are then computed. The ED stores $seskeys_{i+1}$ only while NS stores both $seskeys_i$ and $seskeys_{i+1}$. Before ED sends a frame, the attacker immediately sends to NS a Join Request message she previously eavesdropped on (and not received, hence new to the server). The server computes new session keys $seskeys_{i+2}$ which replace the unconfirmed keys $seskeys_{i+1}$. Then NS stores $seskeys_i$ and $seskeys_{i+2}$ while ED stores $seskeys_{i+1}$. Hence ED and NS do not share the same session keys. More generally, if NS keeps the latest valid session keys and m new sets of keys, the attacker must send successively m new Join Request messages in order to desynchronise NS and ED.

This attack is based on the ability for the attacker to gather multiple and new Join Request messages (i.e., fresh rnd_E values). However if ED sends again the same Join Request message when it does not receive a valid answer from NS, then it is possible to disconnect ED once and for all. Indeed, in such a case, if ED does not receive a valid Join Accept message when it starts a new session (e.g., the attacker sends a "false" Join Accept message before NS), it will keep sending the (same) request which is then continuously discarded by NS since it has already received the message. Hence ED gets unable to connect the back-end network.

[7] This is introduced in version 1.0.2 of the specification and does not appear in version 1.0.1. Also the LoRaWAN specification does not demand the same regarding ED.

Technique used to achieve the attack. In order to get a new Join Request message the attacker can use the technique described in Section 3.3.1.1 aiming at compelling ED to generate multiple Join Request messages. The attacker can gather several such messages and use these anytime later as "desynchronisation ammunition".

Impact. The consequences of this attack against NS are the same as the one against ED: the targeted ED is disconnected from the network. Unaware that NS does not share the same security parameters, it may keep sending uplink frames for quite a long time while NS is unable to process them. Conversely, the frames NS may send cannot be processed by ED.

3.3.3 Lack of Data Integrity (Attack A5)

The LoRaWAN protocol aims at providing data confidentiality and data integrity on the air interface, between ED and NS. However the data exchanged between NS and AS are only encrypted but not integrity protected since NS is the only one to own the key used to compute an authentication tag. Hence AS is not able to verify if a (encrypted) payload has been modified. The specification recommends to implement (at the application level) an integrity protection mechanism. Moreover the specification seems to imply that such a mechanism is in fact optional since "*Network servers are considered as trusted*" ([SLE+16], §6.1.4, p. 32). This is a bold statement. Firstly the threat may not come only from NS (even if it can also be dishonest or compromised). An attacker may target the link (and intermediary servers) between NS and AS. Secondly, it is obvious that encryption only does not provide data integrity, but it may even not be sufficient to guarantee data confidentiality due to the malleability of the operation mode (in particular in the LoRaWAN case with the counter mode).

Goal. The purpose of the attacker is either to modify or to decrypt an encrypted payload. The frame carrying the payload may be sent by ED to AS or sent in the converse direction. Contrary to the attacks described in Sections 3.3.1 and 3.3.2, the attacker here must be able to act on the link between NS and AS. For instance the attacker could target and try to intrude on a (not or poorly protected) MQTT server used to relay data between NS and AS [Lun17].

Attack on data integrity. Data encryption is done in counter mode, therefore it is possible to change the plaintext by flipping bits of the ciphertext. If the content of an encrypted payload or merely the format of the unencrypted content is known, the attacker can replace or alter the data with accuracy. For instance, if ED is a sensor the attacker could change the measurement (temperature, humidity, etc.) sent. If ED is a presence sensor, the attacker may change a (binary) value notifying an intrusion into the opposite value notifying that everything is quiet. If ED is an actuator, the attacker could change a command ordering to close a window into a command ordering to open it. The attacker could also truncate the encrypted payload in order to hide information to AS or ED.

If the recipient is AS, it is not able to detect that the payload is modified since there is no authentication tag. If the recipient is ED, it is not able to detect the modification since the authentication tag is computed by NS after the attacker modifies the frame (in fact ED will validate the frame).

Attack on data confidentiality. The attacker may try to guess the plaintext corresponding to an encrypted frame as follows. She eavesdrops on a frame which payload is of the form $c = k \oplus m$, where k is the keystream, and m the plaintext to recover. She makes a guess m' regarding the

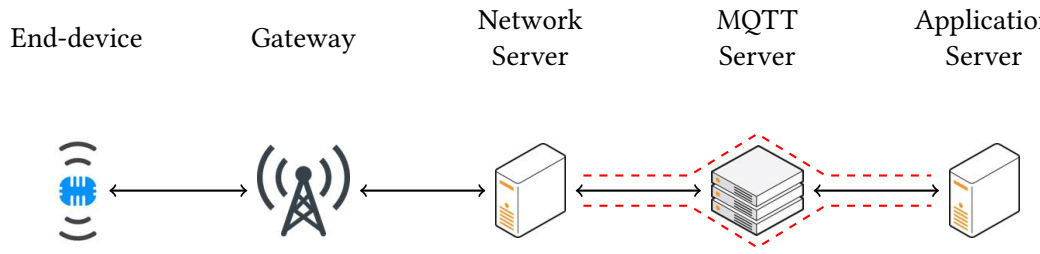


Figure 3.7 – Attack on data integrity. The lack of data integrity between NS and AS (outlined with a red dashed line) enables trivial attacks against ED and AS, by modifying genuine encrypted frames.

plaintext, and chooses a message u from a set of valid applicative messages (e.g., a predefined list of commands shared by ED and AS). The attacker computes $c' = c \oplus (m' \oplus u)$, and sends c' (to the server or to ED). If the guess is correct ($m = m'$) then $c' = c \oplus (u \oplus m') = c \oplus (u \oplus m) = k \oplus u$. Hence the decryption will be correct and the command will likely be completed. Using this kind of "command oracle" attack [AP13; CHVV03; AIES15], the attacker may rely on the expected behaviour of the recipient (either ED or AS) to understand if her guess is correct. If the attacker targets ED, she may do experiments with one such specimen she owns in order to learn first how ED behaves, before acting.

If the recipient is AS, the attacker can make several tries (using the same encrypted payload and frame counter), because AS unlikely verifies the frame counter (since NS does it). On the contrary, if the recipient is ED, the attacker can make one try only, because ED verifies the frame counter and will reject subsequent downlink frames carrying a reused counter.

Attack on data authenticity. If the attacker succeeds in recovering a keystream k it can forge any ciphertext of her choice. Since the uplink counter is verified by NS, likely AS does not check it, and uses the received parameters in order to decrypt the frame.

The attacker can recover the keystream if she knows the corresponding plaintext. If the attacker succeeds in decrypting data through the "command oracle" attack described above, she also gets the corresponding keystream (partially or totally). Then she can use it to forge encrypted payloads intended to AS.

This attack is not due to a lack of protection of some intermediary server (between NS and AS, such as an MQTT server). If the application frames were duly protected, the worst an attacker could do would be to delete frames. However, since LoRaWAN does not provide end-to-end integrity protection between ED and AS, it is possible to deceive both of them.

Table 3.1 summarises the attacks we have found against LoRaWAN 1.0.

Table 3.1 – Attacks against LoRaWAN 1.0. n_{ja} is the number of Join Accept messages usable by the attacker. n_{jr} is the number of Join Request messages usable by the attacker. m is the number of new session keys sets stored by NS.

Attack		Complexity (# Join message)	Probability of success	Impact
(A1)	Replay or decrypt (ED, Section 3.3.1.1)	$\frac{2^{16}}{n_{ja}}$	$\simeq 1$	Downlink frame replay. Uplink frame decryption.
(A2)	Replay or decrypt, method 1 (NS, Section 3.3.1.2)	n_{jr}	$\simeq \frac{n_{jr}}{2^{24}}$	Uplink frame replay. Downlink frame decryption.
	Replay or decrypt, method 2 (NS, Section 3.3.1.2)	$\simeq 2^{13}(n_{jr}+1)$	$\frac{1}{2}$	Uplink frame replay. Downlink frame decryption.
(A3)	Desynchronisation (ED, Section 3.3.2.1)	1	1	ED desynchronisation
(A4)	Desynchronisation (NS, Section 3.3.2.2)	m	1	ED desynchronisation
(A5)	Data integrity (Section 3.3.3)	-	1	Uplink frame replay, forgery, decryption. Downlink frame forgery.

3.4 Recommendations for LoRaWAN 1.0

Hereinafter, we propose practical recommendations aiming at thwarting the attacks described in Section 3.3. Obviously, one could merely advise the replacement of the protocol with a more secure protocol. Yet, we take into account that there are already many deployed LoRaWAN networks. Hence, as an additional constraint, we aim at proposing improvements that can solve the issues as best as possible while at the same time remaining compliant with the specification, and keeping the interoperability between patched and unmodified equipment. All the attacks can be mitigated if both ED and NS are corrected. Otherwise, the attacks targeting an unmodified equipment remain possible.

3.4.1 Practical Implementation

Against attack A5: implement end-to-end data integrity between ED and AS. This must be done at the application level. In addition, AS must not blindly trust NS and should verify every security parameter it receives. In particular, if AS receives the application session key K_a^e from NS, it should verify that the key is fresh (not reused). Similarly AS must keep track of the frame counters (both uplink and downlink counters) in order to avoid frame replays.

Against attack A4: verify that the session keys are shared. We suggest to implement it in the following way. Straight after the key exchange is done, NS must send a so-called *DevStatusReq* command and verify (authentication tag) the *DevStatusAns* response from ED, or verify, if it comes earlier, the first frame sent by ED. The lack of response must be read into this as an issue (ED or NS under attack).

In addition NS must keep all sets of session keys from the last valid one up to the latest

computed one. When NS receives an uplink frame (carrying a *DevStatusAns* response, or another uplink frame), it checks the authentication tag with all keys, starting from the latest. If the keys that match with the authentication tag belong to one of the yet unapproved sets, then NS keeps this set of session keys only and drops all the others. This set becomes then the last valid one.

Against attack A3: verify that the received Join Accept message corresponds to the sent Join Request message. We recommend to compute the $DevAddr$ parameter in the following way. Let $NwkAddr$ be the least 25 significant bits. $NwkAddr$ is computed as $NwkAddr = \mathsf{H}(rnd_E, rnd_N, id_E)$ where H is a collision-resistant function.

Alternatively, ED can perform a session key confirmation with NS.

Against attack A2: generate rnd_N values with no repetition. A counter must be used to produce the rnd_N values. The counter must not overlap, and one different counter must be used for each ED in order not to artificially lower the number of sessions per ED.

Against attack A1: detect a replay of rnd_N values. It may be implemented using computationally and memory efficient techniques such as Bloom filters [Blo70; DM04]. However the rnd_N parameter being turned into a counter, it is enough for ED to store the last received rnd_N value in order to detect a replay.

3.4.2 Changes in the LoRaWAN Specifications

Version 1.0. LoRa Alliance, in charge of developing and promoting the LoRaWAN specification, has been informed of the attacks described in Section 3.3. A document recommending changes to be implemented in LoRaWAN 1.0 has been published [LoR18b]. Two recommendations correspond to ours: the rnd_N parameter is turned into a counter, and a session key confirmation is done (by ED). Yet, the key confirmation message can also transport application data.

The document recommends also to turn the rnd_E parameter into a counter. This aims at preventing a replay of Join Request messages (which enables attacks A2 and A4). According to us, this is hazardous. Indeed, if the rnd_E counter does not overlap, then an attacker can compel ED to use all possible counter values (simply by responding an invalid Join Accept message to its Join Request messages). Eventually, ED ends up being unable to connect the back-end network, and blocked once and for all.

Furthermore, end-to-end data integrity between ED and AS remains unenforced.

Version 1.1. As presented in Section 3.5, this version differs from version 1.0 in several ways. Nonetheless, the rnd_N counter, and the session key confirmation (done by ED) have been maintained.

3.5 Protocol LoRaWAN 1.1

3.5.1 Architecture

LoRaWAN version 1.1 has been published in 2017 [Sor17]. It aims at replacing version 1.0 and at providing corrections to this previous release. It introduces also a third entity called Join Server (JS), turning the original 2-party protocol (between ED and NS) into a 3-party protocol (between ED, NS and JS). JS is connected to NS, and, possibly, to AS. The AS servers remain

present in the architecture. The role of the latter servers, as in version 1.0, is merely to use the session key computed by the other entities, and to exchange encrypted frames with ED. Figure 3.8 depicts the architecture of a typical LoRaWAN network in version 1.1.

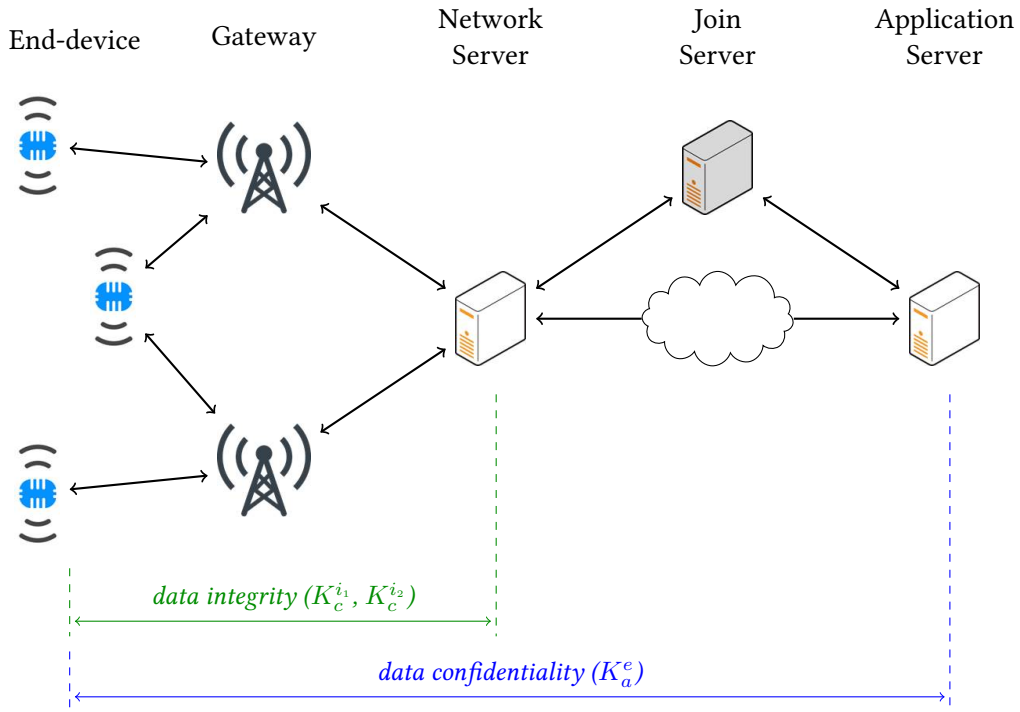


Figure 3.8 – LoRaWAN network in version 1.1. Data exchanged between ED and NS are encrypted with K_c^e. Data exchanged between ED and AS are encrypted with K_a^e.

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 (it inherits from version 1.0). The Rejoin type 1 procedure is an "emergency" method aiming at reconnecting ED in case of total loss of the cryptographic context by 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 the remaining of this chapter we consider the method likely the most used to execute the protocol, that is the Join procedure.

3.5.2 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. All the cryptographic operations are based on the AES block cipher.

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. Figure 3.9 depicts a session establishment in version 1.1.

The Join Request message sent by ED carries three main parameters: JS' identifier id_J (64 bits), ED's identifier id_E (64 bits), the current ED's counter cnt_E (16 bits). These parameters are protected with a 32-bit MAC tag computed with AES-CMAC keyed with the master key MK_1. 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).

$$\text{Join Request} = id_J \| id_E \| cnt_E \| \tau_E$$

with

$$\tau_E = \mathsf{MAC}(MK_1, id_J \| id_E \| cnt_E).$$

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 (24 bits), NS' identifier id_N (24 bits), and other parameters $prms$ (such as radio parameters). Since the $prms$ parameters are not relevant to the remaining of this chapter, we skip their description and refer the interested reader to the specification [Sor17]. The message 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. This aims at forbidding a replay of previous Join Accept messages to ED (which the previous version of the protocol, LoRaWAN 1.0, is subject to as shown in Section 3.3.1.1). 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 MAC tag and the cnt_J counter.

$$\text{Join Accept} = \mathsf{AES}^{-1}(MK_1, cnt_J \| id_N \| prms \| \tau_J)$$

with

$$\begin{aligned} \tau_J &= \mathsf{MAC}(MK_3, id_J \| cnt_E \| cnt_J \| id_N \| prms) \\ MK_3 &= \mathsf{KDF}_{mk}(MK_1, id_E). \end{aligned}$$

The KDF_{mk} function corresponds to

$$\mathsf{KDF}_{mk}(K, x) = \mathsf{AES}(K, \texttt{0x06} \| x \| \texttt{0x00000000000000}).$$

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.

3.5.3 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:

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

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,

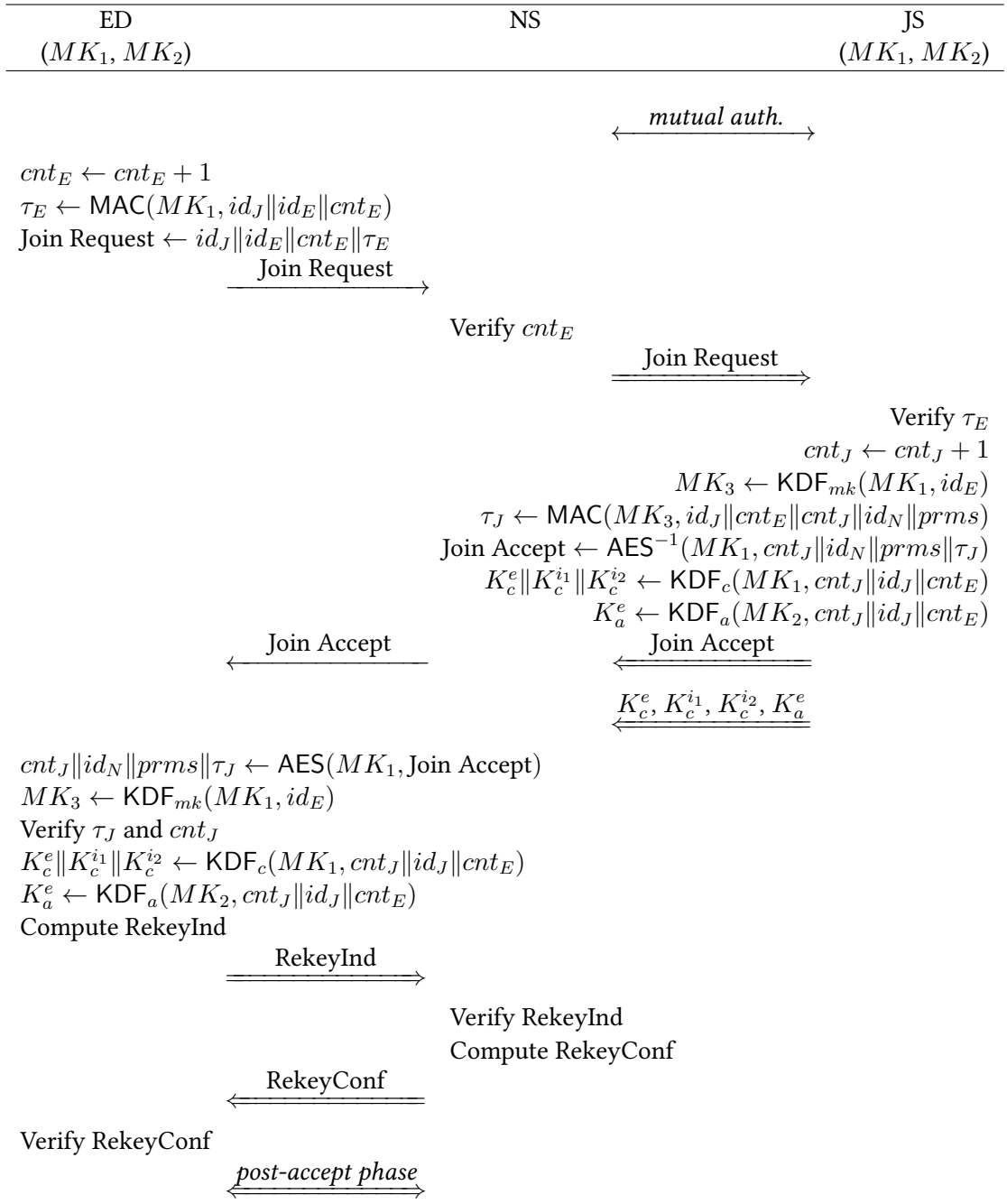


Figure 3.9 – Correct execution of LoRaWAN 1.1. Double line arrows indicate the use of secure channel keys. There are two secure channels: ED-NS (LoRaWAN), and NS-JS (additional protocol).

only known to JS and AS [Yeg17]). The KDF_c function, on input a key K and a value x, outputs

the following three values

$$\begin{cases} K_c^{i_1} &= \mathsf{AES}(K, \texttt{0x01}\|x\|\texttt{0x0000}) \\ K_c^{i_2} &= \mathsf{AES}(K, \texttt{0x03}\|x\|\texttt{0x0000}) \\ K_c^{e} &= \mathsf{AES}(K, \texttt{0x04}\|x\|\texttt{0x0000}) \end{cases}$$

The function KDF_a is defined as

$$\mathsf{KDF}_a(K, x) = \mathsf{AES}(K, \texttt{0x02}\|x\|\texttt{0x0000}).$$

3.5.4 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. In addition, irrespectively of the frame type, part of the header is encrypted with K_c^e. Three different frame counters are used. They are 32-bit each. One frame counter is used in the uplink direction. In the downlink direction, AS and NS use a different frame counter.

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 32-bit 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.

The format of an encrypted frame is the same as in version 1.0:

$$hdr\|ctext\|\tau.$$

3.6 Vulnerabilities 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.

3.6.1 Size of the Counters

Key points. The counters cnt_E, cnt_J are respectively 16-bit and 24-bit long. It is likely that such short counters 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.

Technique 1. 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 be stored into ED (as in the previous version, where cnt_E is a pseudo-random value). Moreover it has been shown in Section 3.3.1.1 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 [Wor], 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.

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 (especially if NS did not lose this context).

Technique 2. 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 24-bit long, whereas cnt_E is 16-bit 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.

3.6.2 Size of the MAC Tags

As in version 1.0, the MAC's output is 32-bit long in LoRaWAN 1.1. Hence, MAC forgeries are worth considering, and, in combination with the fact that data encryption is done in CTR mode, so are attacks against data integrity.

The duration of such forgeries is higher if the attacker acts on the air interface (in particular if she targets ED) than if she is able to act in the back-end network.

3.6.3 Known Encryption Keystream

Key points. 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 (static value described in the specification). 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.

Technique used. 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. She 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 $n = \texttt{ADR\_ACK\_LIMIT}$ is at most 2^{15}. Nonetheless, the default settings [Wor] in several geographical areas (e.g., USA, Europe, China) demand that $n = 64$. Therefore, the attacker has at her disposal between 64 and 2^{15} frames (in either direction). Yet, when ED reaches this limit, it must initiate a new key exchange, which provides to the attacker a new batch of n messages in the downlink and uplink directions.

Impact. The encrypted payload of the RekeyInd and RekeyConf commands is 8-bit long. Therefore, a successful attacker gets a 8-bit encryption keystream. The attacker can use it to encrypt other 8-bit long commands. The following commands can be of interest to the attacker: ADRParamSetupReq, DutyCycleReq. These commands are intended to ED. ADRParamSetupReq is used to change the value of the $n = \texttt{ADR\_ACK\_LIMIT}$ parameter. The attacker can then set this parameter at its highest value in order to favour subsequent attacks. DutyCycleReq is used to change the transmission rate of the uplink frames (including the Join Request frames). Hence, increasing this rate is a way to passively exhaust the 16-bit cnt_E counter. Moreover, the LoRaWAN 1.1 specification allows defining proprietary commands, which could also be exploited by the attacker to target ED or NS.

3.6.4 Downgrade Attack

Key points. 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 (see Section 3.3). Therefore, a current deployment of LoRaWAN 1.1 may inherit the flaws of the previous version.

Scenario 1. In this scenario, ED in version 1.1 faces first NS in version 1.0. Then, at some point, NS is upgraded to version 1.1. Both ED and NS execute LoRaWAN 1.1. If an attacker eavesdropped on a Join Accept message computed in accordance with version 1.0 (when NS was applying that version), she can send it to the upgraded ED. Then ED accepts the message and falls back to version 1.0.

Scenario 2. In this scenario, ED in version 1.0 is deployed, and communicates with NS in version 1.0. At some point it is upgraded to version 1.1 (as well as NS). Since ED is already provisioned and registered on the back-end side, it is possible that ED keep the same identifier id_E. If, in addition, the same master key used in version 1.0 is used as the MK_1 master key in version 1.1, then several attacks are possible against the (upgraded) ED.

Impact. Scenario 1 enables the desynchronisation attack (see Section 3.3.2.1). Scenario 2 enables the “replay or decrypt” attack against ED (see Section 3.3.1.1). Indeed, the reuse of

session keys and encryption keystreams can be obtained when the 16-bit monotonically increasing counter cnt_E that ED uses in version 1.1 meets same values as the 16-bit pseudo-random parameter rnd_E used by ED in version 1.0. Moreover the desynchronisation attack is also possible with scenario 2.

It may be possible that, when executing version 1.0, ED implements also the recommendations published by LoRa Alliance [LoR18b] which aim at strengthening the security of version 1.0. With respect to the desynchronistion attack, this document recommends that ED initiate a new key exchange if it does not receive a response from the back-end network after `ADR_ACK_LIMIT` uplink frames. We observe that, even in such a case, the desynchronisation attack can be detrimental to the network availability. Indeed the regular behaviour of ED may imply sending one frame per hour or even per day. Since $\texttt{ADR\_ACK\_LIMIT} = 64$, several days or weeks may elapse without the back-end network being able to communicate with ED. Likewise, all the data sent meanwhile by ED will be lost, and unusable by the network.

With respect to scenario 2, if ED enhances version 1.0 with the LoRa Alliance recommendations then the "replay or decrypt" attack (see Section 3.3.1) is mitigated, because in such a case the pseudo-random values involved in the session key derivation are replaced with monotonically increasing counters.

3.6.5 Lack of Data integrity

Key points. 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 [Yeg17] 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 [Lun17].

Impact. 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" (see Section 3.3.3). 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 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]

3.6.6 Reuse of an Application Session Key

Key points. 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.

[8] Rupprecht, Kohls, Holz, and Pöpper [RKHP19] have shown (though in another context) the consequences of the combination of encryption in CTR mode and lack of data integrity.

Technique used. The attacker acts on a weakly protected intermediary point between NS and AS where she 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.

Impact. 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 [MWES06].

Another issue arises also. The application frame then 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.

3.6.7 Malicious or Corrupted NS

Key points. A malicious or corrupted NS which receives the encrypted application keys K_e^a from JS can apply the scenario described in Section 3.6.6. It can also possibly do more.

A companion specification [Yeg17] states that the communication and application master keys MK_1 and MK_2 must be stored at JS which is then in charge of doing the key exchange with ED. At the same time, the specification version 1.1 [Sor17] claims that the communication master key may be given to the communication provider. Regarding the application master key, the specification does not allow nor forbid from surrendering the MK_2 key to AS. Therefore we provide the following observation. Let us assume that AS owns MK_2 and computes the application session key K_a^e. This key is computed from the two variable parameters cnt_E and cnt_J. We acknowledge that cnt_J must be checked by JS. Hence, AS may receive that parameter from JS. Nonetheless, per specification, NS may also be in charge of relaying to AS parameters received from JS (e.g., the encrypted key K_a^e). Therefore, it is conceivable that JS rely upon NS in order to send cnt_J to AS.

Technique used. If cnt_E and cnt_J are received by AS from NS (and if AS does not keep track of these parameters), a malicious or corrupted NS can choose any pair of values, and compel AS to compute and use past, current or future application session keys. Recall that NS chooses also the $DevAddr$ parameter which is involved in the encryption keystream (S_i) computation.

Impact. If AS reuses a past key K_a^e, the attacker can replay to AS old frames, and also attempt frame decryption. If AS uses a future session key to encrypt an application frame, the attacker can collect (e.g., on the air interface) and store these frames in order to use them some time later against ED. The malicious NS can act so against any AS it is legitimately allowed to communicate with.

The scenarios described in Sections 3.6.6 and 3.6.7 imply stringent assumptions (intruding an intermediary server or NS). Our purpose is not to elaborate on the difficulty of achieving

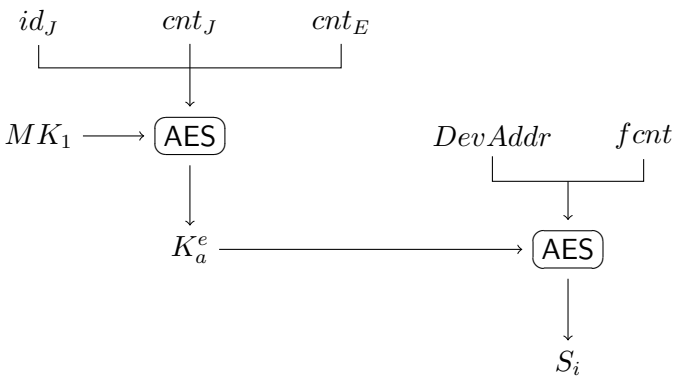


Figure 3.10 – Application session key derivation (K_a^e) and encryption keystream computation (S_i) in LoRaWAN 1.1

the latter but rather to point out some issues left unattended in the specification due to its lack of clarity and the limited security perimeter it covers. That is, under the same scenario, and with the same capabilities (e.g., breaking into an MQTT or NS server), an attacker can be more detrimental to LoRaWAN 1.1 than to a state-of-the-art protocol. In particular, the LoRaWAN specification seemingly considers the protocol as involving two parties only (ED and the back-end network), and does not make clear the security properties it is supposed to guarantee, nor the powers of the adversary it aims at defending against. 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.

In Chapter 5, we devise a security model that addresses the LoRaWAN protocol in its 3-party setting (ED, NS-AS co-localised, JS), and allows capturing the security properties it should guarantee. Then we use this security model in order to provide a security proof of a LoRaWAN 1.1 protocol modified in order to mitigate the aforementioned flaws.

3.7 Recommendations for LoRaWAN 1.1

In this section, we present several recommendations aiming at mitigating the attack scenarios described in Section 3.6. The recommendations take as most as possible into account the need to keep the interoperability between a patched equipment and an unmodified equipment.

R1 – Exhaustion of counter. Against the scenario described in Section 3.6.1 based on the size of the counters, we recommend that ED store enough id_J values such that exhausting counter cnt_E be not possible.

R2 – MAC tag forgery. Thwarting a MAC tag forgery (Section 3.6.2) is not possible without increasing the size of the tag. We recommend the latter even though this forbids two entities (patched and unmodified) from being able to communicate.

R3 – Known encryption keystream. In order to preclude the scenario described in Section 3.6.3, we recommend that ED and NS perform a proper session key confirmation prior to

transmitting any application or command frame. Alternatively, the same procedure as the one described in LoRaWAN 1.1 can be applied, but based on a reduced number of RekeyInd/RekeyConf commands in order to lower the extent of the attack.

R4 – Downgrade attack. Against the attack described in Section 3.6.4, we suggest three distinct countermeasures. Firstly, ED in version 1.1 must refuse to fallback to version 1.0 when it faces NS implementing only this version. According to us, such a circumstance should be rare since NS will likely implement both versions. Secondly, ED in version 1.1 can execute version 1.0 following the LoRa Alliance recommendations [LoR18b]. Yet, as indicated in Section 3.4.2, this allows (against this modified version 1.0) the attack aiming at exhausting ED's counter (see Section 3.6.1). Thirdly, NS in version 1.0 can compute the Join Accept message the same way as in version 1.1. This allows ED to verify that the message it receives is indeed a response to the Join Request message it has sent.

R5 – Lack of data integrity. The attack due to the lack of end-to-end data integrity between ED and AS (Section 3.6.5) can be mitigated the same way as for version 1.0. That is, implementing data integrity at the application level between these two parties, and keeping track (by AS) of the frame counter.

R6 – Reuse of an application session key & malicious or corrupted NS. Likewise, in order to thwart the attack scenarios described in Sections 3.6.6 and 3.6.7, AS must keep track of all parameters used in the application session key computation as well as in a frame encryption and decryption. This includes the counters cnt_E and cnt_J, the uplink and downlink frame counters, and the application session key (when received from NS). This countermeasure can be implemented using computationally and memory efficient techniques such as Bloom filters [Blo70; DM04].

3.8 Other Analyses

In this section we recall independent analyses on LoRaWAN 1.0 and 1.1 that have been published while our papers [AF18b; CF19] were under submission or after they have been accepted.

3.8.1 On LoRaWAN 1.0

Few analyses on LoRaWAN have been done. Most of the reviews deal with technical consideration such as the network management (secret keys storage, etc.) and generic attacks (e.g., hardware attacks, web attacks) unrelated to the LoRaWAN protocol itself.

In contrast to these works, we have provided an extensive analysis of the protocol and show that it suffers from several weaknesses. We have presented new attacks and for each of them we have provided a precise description of its goal, its implementation, the technical means used, and the tangible consequences. In addition we have described attacks targeting either ED or NS. Our attacks do not lean on strong assumptions such as the ability to monitor ED or NS. Likewise the attacks do not entail a physical access to the targeted equipment and are independent from the means used to protect secret values (e.g., using a tamper resistant module such as a Secure Element). The attacks, due to the protocol weaknesses, do not lean on potential implementation or hardware bugs, and are likely to be successful against any equipment implementing LoRaWAN 1.0.

Lifchitz [Lif16] notes that, since the rnd_E and rnd_N parameters are pseudo-random, a reuse is possible due to the birthday paradox. Hence, under the strong assumption that the rnd_E value is "*forced*" (i.e., ED controlled by an attacker), a keystream reuse happens with high probability after $\sqrt{2\ln(2) \times 11 \times 2^{24}} \simeq 16{,}000$ sessions, or 22 hours if a key exchange is done in 5 seconds. According to us, if both rnd_E and rnd_N values repeat, this leads to a *session keys* reuse. In order for a *keystream* to repeat, it is necessary for the $DevAddr$ parameter to be reused as well. Moreover the figure of 22 hours corresponds to a continuous series of key exchanges without any intermediary application frame being sent. The purpose of such an attack is likely to manipulate (decrypt, replay) such frames. Therefore, the time needed to collect these frames must be taken into account, and, in all likelihood, the duration of this attack is higher.

Furthermore it is unclear where the numbers come from. We may assume that NS keeps track of 10 rnd_E values, and the attacker uses 11 different Join Request messages, randomly choosing one at each try. However NS unlikely accepts every such message since the same message (hence the same rnd_E value) is picked after roughly 4 tries. Hence the number of sessions needed to get a reuse of both rnd_E and rnd_N values is higher than the provided number 16,000, as well as the duration of the attack. The number of 10 stored values seems also to be implementation specific. Yet we cannot claim this is the genuine purpose of [Lif16].

Lifchitz's analysis is made on version 1R0 of the protocol [SLE+15] published in January 2015. We observe that this attack is unlikely successful against an NS implementing version 1.0.2 (the current 1.0 version) published in July 2016. Indeed, according to the specification, NS must receive a valid uplink frame protected by the new security parameters before dropping the current ones and using the new ones. The attack described by Lifchitz leads to the computation of the same session keys two different times. Yet, with high probability, these keys are fresh (i.e., never used previously by NS with a legitimate ED) because the attacker has no control on the rnd_N parameter. This means that the attacker has to forge a valid uplink frame if she wants to compel NS to use these keys. That is, the attacker must forge a valid 32-bit authentication tag.

Lifchitz indicates also that an alternative way to attack ED is to replay to NS a previous Join Request message (because the server keeps track of a "*certain number of* $[rnd_E]$ *values*"), leading to ED being disconnected from the network.

Finally, Lifchitz observes that the encryption scheme does not hide the length of the plaintext which may help an attacker to deduce the underlying data.

Miller [Mil16b; Mil16a] proposes a denial of service attack against NS by flooding the server. He notes also the possibility to replay Join Request messages. Moreover, since the application frames are protected with a 32-bit authentication tag, forgery attempts may be tried mainly against NS, according to Miller.

Tomasin, Zulian, and Vangelista [TZV17] indicate that ED may be precluded from joining the network after a certain number of sessions, depending on the NS' behaviour. This may happen either "naturally" (if NS keeps track of all received rnd_E values), or due to an attack (if NS decides to exclude ED which it repeatedly receives Join Request messages replays from – the frames being replayed sent by an attacker). We observe that if the rnd_E value repeats there is a more detrimental attack than a DoS, namely the "replay or decrypt" attack described in Section 3.3.1.1.

Moreover Tomasin et al. recommend to use a 16-bit counter for rnd_E instead of a pseudo-random value, and to increment the counter only when a (valid) Join Accept message is received. This aims at ensuring that the rnd_E counter is shared by ED and NS. According to us, this is incorrect because it is possible to replay any Join Accept message. In addition, this recommen-

dation leads to another attack which allows disconnecting ED from the network once and for all. This attack is of the same kind as the one described in Section 3.3.2.2. The attacker does the following: when ED sends a Join Request message, she forbids (once only) ED from receiving the Join Accept message sent by NS. Hence ED reuses the same rnd_E value in subsequent Join Request messages. And these messages are then discarded by NS since they carry an invalid (i.e., reused) rnd_E value. Therefore ED is stuck since it keeps using the same rnd_E value (which is continuously discarded by NS).

Another issue may also happen if NS keeps a strict synchronisation with ED. Since an attacker can easily replay any Join Accept messages, she can respond to ED (when it sends a Join Request message) before NS, while forbidding the latter from receiving ED's message. Hence ED increments its rnd_E counter while NS does not. The more the attacker repeats this procedure the greater the difference between both counters. If NS accepts a Join Request message only if the difference between the two counters is exactly 1 then ED's message is likely rejected.

Finally, Tomasin et al. show also that it is possible to make the distribution of the random bit generator output produced by ED (hence the distribution of the rnd_E values) deviate by influencing on the radio signal strength, because the specification suggests to compute the rnd_E values from radio measurements.[9]

Yang [Yan17] observes that it is possible to replay previous frames and to decrypt frames if some security parameters are reused (namely the frame counter, and encryption keytsream). This can be done if the frame counter wraps around. According to the author, the latter may happen if ED is reset (in the case of the ABP mode where ED uses always the same session keys), or if it sends an amount of frames that exceeds the size of the frame counter. However the specification forbids ED from using twice the same frame counter value (with the same session keys). Moreover Yang's attacker targets NS, and, per specification, NS must discard any frame replay. Hence, the feasibility of such attacks seems unlikely according to us.

Yang indicates also that the lack of data integrity between NS and AS allows an attacker to modify the plaintext by modifying the encrypted payload (due to the encryption in counter mode).

ED can ask NS to acknowledge the reception of an uplink frame. But the frame sent by NS in response is not related to the uplink frame it is supposed to acknowledge. Hence, Yang observes that this can be used to deceive ED. An attacker can eavesdrop on such an acknowledgement response, forbid ED from receiving it, and send it later on to ED to make the latter believe that a subsequent uplink frame (demanding to be acknowledged) has been received by NS whereas it is not the case.

3.8.2 On LoRaWAN 1.1

In contrast to previous analyses, we have addressed in our analysis the new 3-party aspect of the protocol introduced in its latest version. In addition, we have described new vulnerabilities impairing the security of LoRaWAN 1.1, and proposed several recommendations aiming at mitigating the latter. Finally, we have used a provable security approach that enables to shed light on the security requirements implied by this 3-party protocol. We detail this contribution in Chapter 5, Sections 5.3 and 5.4.

Butun, Pereira, and Gidlund [BPG18] 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

[9] Tomasin acknowledge our remarks [Tom19].

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 [AVTP+17] 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 [Sor17]), 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.

In LoRaWAN 1.1, the MAC tag of an uplink frame is computed with two session keys: the first 16 bits with $K_c^{i_1}$, the last 16 bits with $K_c^{i_2}$ (cf. Section 3.5.4). This computation aims at allowing an intermediate NS (called *forwarding* NS) to verify an uplink frame in a roaming configuration. Consequently Stöhr [Stö17] devises the following attack scenario. If the attacker is able to know independently whether either 16-bit string of the MAC tag is correct, she can forge a valid MAC tag with at most 2×2^{16} trials instead of 2^{32}. The attacker can lean on the behaviour of the back-end network to achieve her attack (e.g., the NS servers may return different error codes, or respond faster depending on the validity of the two pieces of the MAC tag).

This scenario highlights that the size of the MAC tags may not be long enough in order to provide enough resistance against forgeries.

Dönmez and Nigussie [DN18b] 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 16-bit monotonically increasing counter that ED uses in version 1.1 meets same values as the 16-bit 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 and Nigussie to fall back to version 1.0), and come essentially to the same conclusion.

Dönmez and Nigussie [DN18a] 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 [DN18b]) 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 and Nigussie 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*" ([Sor17], §6.1.1.3). Dönmez and Nigussie 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 and Nigussie 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 [Sor17] (cf. §6.1.2.3) and a companion document [Yeg17] (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 and Nigussie 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 and Nigussie 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 3.6.7 how a malicious (or corrupted) NS can compel AS to reuse a previous session key, without JS' counter wrapping around.

Finally, Dönmez and Nigussie 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 [KS17]. 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 and Song nor Dönmez and Nigussie explain how to maintain this essential synchronisation between ED and the back-end network.[10]

[10]Dönmez and Nigussie acknowledge our remarks [Dön19]. Regarding Kim and Song's proposal, Dönmez and Nigussie acknowledge that a mechanism aiming at ensuring synchronisation remains to be defined.

Remark. In Chapter 6, we describe an authenticated key exchange protocol in the symmetric-key setting that provides forward secrecy, and explicitly deals with the synchronisation issue.

In turn, Han and Wang [HW18] 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 [EBPG18] 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 that there may still exist vulnerabilities not found due to the limitations of the model they employ. It could be advantageous to do an in-depth analysis with such a verification tool based on a more extensive model.

4 Analysis of SCP02

SCP02 is a symmetric-key protocol implemented in a specific type of constrained devices, namely smart cards. It is used by transport companies, in the banking world and by mobile network operators (UICC/SIM cards) to transmit sensitive data through a secure tunnel. Among a wider set of symmetric-key protocols promoted by GlobalPlatform, SCP02 is likely the most used by the industry of smart cards.

In Chapter 3, we have seen how to exploit flaws that weaken the key exchange phase in LoRaWAN 1.0. In this chapter, we consider the secure channel established after the key exchange phase in SCP02, and show how to perform a padding oracle attack against this protocol. This attack allows to efficiently retrieve data protected within the secure channel.

We have implemented the attack in an experimental setting, and we present the results of our experiments done with 10 models of smart cards produced by six different card manufacturers. This shows that the attack is fully practical in our experimental setting. Given that billions SIM cards are produced every year, the number of affected cards, although difficult to estimate, is potentially high. To the best of our knowledge, this is the first successful attack against the SCP02 protocol.

The results of this chapter have been published in [AF18a].

Contents

4.1 Introduction

GlobalPlatform is an organisation that aims at defining technical mechanisms related to the chip technology (e.g., smart cards, application processors, SD cards, USB tokens, secure elements), used to securely add and remove applications, and related parameters, into the chips. This initiative aims also at facilitating the interoperable deployment and management of applications on these types of device, regardless of the manufacturer, as well as "*promot*[ing] *a global infrastructure for smart card implementation across multiple industries*" [Glo18].

Several Secure Channel Protocols (SCP) are specified by GlobalPlatform. Most of them are based on symmetric-key cryptography (e.g., SCP01, SCP02, SCP03, SCP81). Regarding the protocols status, SCP01 (based on the DES algorithm) is now deprecated. SCP02 (based on 3DES) is currently the most deployed symmetric-key based SCP protocol[1], while the use of SCP03 (based on AES) seems to be less widespread.[2]

SCP02 is a protocol aiming at establishing a secure channel between a card and an "*off-card entity*". The main purpose of the protocol (yet not the only one) is the management of a (remote) card. Through the secure channel established with SCP02, applets, files, secret data (e.g., encryption keys, PIN codes), etc., may be transmitted and stored into the card.

According to GlobalPlatform, SCP02 is used by transport companies, in the banking world and by mobile network operators (UICC/SIM cards). According to GSMA [GSM19], in 2018 there are over 5 billion unique subscribers to mobile services, and 5.6 billion of SIM cards used worldwide as estimated by the SIMalliance [SIM18].

A typical usage of the SCP02 protocol is the management of a specific application storing user credentials (e.g., for payment or transport transactions) located in the UICC card that is plugged into a smartphone. Depending on the context, an additional security protocol (e.g., TLS [DR08] or SCP80 [ETS17]) may be used together with SCP02. For instance, in order for a remote server to send SCP02 commands to the UICC, a SCP02 channel is established between the remote server and the UICC through the intermediary of an application on the smartphone. In addition a second secure channel is established between the remote server and the application on the smartphone (e.g., with TLS or SCP80). This second security layer embeds the SCP02 commands sent by the server. The data exchanged between the server and the application are then protected with the two security layers, whereas the data exchanged between the application on the smartphone and the UICC are protected with SCP02 only.

4.2 The SCP02 Protocol

SCP02 aims at establishing a secure tunnel between an "*off-card entity*" and a card [Glo18]. For simplicity, we will use the term "server" to refer to the party involved in the communication with the card. SCP02 is based on symmetric-key algorithms. The card and the server share one or several sets of symmetric keys. A set is made of three keys (which value may be equal). From that set, session keys are computed each time a new channel is established. The card manages a sequence counter related to a given keyset. Its initial value is 0 and incremented after each successful session (established with that keyset). Once the sequence counter has reached its maximum value, the card must not start anymore a session with the corresponding keyset.

Since the operations done during the key exchange phase are not significant for the remainder of this chapter, we skip the corresponding description, and refer to the SCP02 specification [Glo18].

[1] Last release: March 2018 [Glo18].
[2] Last release: July 2014 [Glo14].

Depending on the security level negotiated during the key exchange, the commands sent by the server and the responses sent by the card are encrypted and protected with a MAC tag. Regarding the commands, the lowest security level is data integrity. Moreover data encryption solely is not allowed. In the remainder of the chapter we consider that the commands are encrypted and MAC-ed.

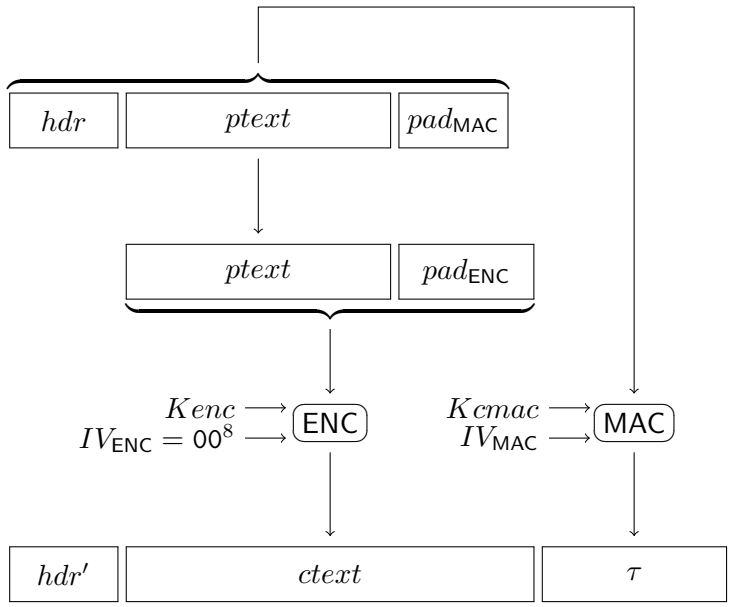


Figure 4.1 – Encryption and MAC computation of a command data with SCP02

Figure 4.1 depicts the data encryption and MAC computation of a command. Data encryption is done with 3DES in CBC mode with a null IV [Int17]. Prior to encryption the plaintext $ptext$ is (always) padded with a fixed string of bytes according to the method described in [Int11]: a byte equal to 80 is appended to the plaintext, then as many null bytes as necessary (possibly none) are added in order to get a string which length is a multiple of a DES block [Int17].

The 8-byte MAC tag τ is computed on the command header hdr, the plaintext, and a padding data pad_{MAC}. The ISO 9797-1 MAC algorithm 3, also known as "retail MAC", is used [Int11]. It consists in first applying DES-CBC-MAC to the input, and then finalising the calculation with one call to 3DES. The value of the first IV for the MAC computation is usually 00^8. The IV used for the next command is equal to the MAC tag computed on the previous command. The ciphertext $ctext$ and the MAC tag τ become then the data field of the server's command.

Upon reception of an encrypted command, the card does the reverse operations. First it decrypts $ctext$. Then it searches for a valid padding data pad_{ENC} and removes it. Finally, the card recomputes a MAC tag and compares it with the tag carried in the encrypted command (the genuine header hdr can be retrieved from the header hdr' of the encrypted command).

4.3 The Generic Padding Oracle Attack

The padding oracle attack is based on the fact that a device behaves differently depending on the correctness of the (encryption) padding data. From that differentiated behaviour, the attacker tries to get some information (e.g., some bits or bytes of plaintext). That difference may be based on the nature of the response (presence or absence of the response, type: e.g.,

"regular" or error message, value, etc.) or on the duration of the operations performed (or not) by the device. Regarding the symmetric-key encryption case, the whole decryption procedure usually includes (among other possible operations) the MAC verification, and the padding extraction and verification. It is commonly recommended to provide data authenticity and confidentiality by applying the so-called Encrypt-then-MAC (EtM) paradigm [Kra01; BN08].[3] Nonetheless some cryptographic mechanisms apply other methods (e.g., MAC-then-Encrypt in TLS 1.2 [DR08], Encrypt-and-MAC (E&M) in SSH [YL06c; YL06a; YL06d; YL06b]) but also in SCP02.

Therefore, if a padding data is used during the encryption process, and the padding data must be, during the decryption procedure, verified *prior* to the MAC computation, then it may be possible to perform an attack aiming at retrieving sensitive data. If one follows the MtE or the E&M method (as in SCP02), the whole decryption procedure involves (usually) three main steps:

1. the evaluation of the decryption function on the encrypted data,
2. the extraction and verification of the padding data,
3. the computation of the MAC tag on the remaining decrypted data.

Once the ciphertext is decrypted, either the padding data is valid and can be removed, and the MAC computation can be done, or the padding data is invalid and the MAC tag cannot be computed (at least on the genuine data).

Let us illustrate the attack with an example. For the sake of clarity, we use the specifics of SCP02, namely the encryption function (and the corresponding block size), and the padding scheme. Let C be the last encrypted block carried in a protected command, and let V be the block used as IV during the encryption operation that yields C (V denotes either the null IV if the command carries one encrypted block only, or the previous encrypted block if the command carries two or more encrypted blocks). Let $b_0|\cdots|b_5$ be 6-byte plaintext data corresponding to C. In SCP02, the encryption is done with 3DES in CBC mode. Since the plaintext length is smaller than 8 bytes a padding data is appended, and this yields $B = b_0|\cdots|b_5|80|00$. The encryption process outputs

$$\begin{aligned} C &= \mathsf{ENC}(V \oplus B) \\ &= \mathsf{ENC}((v_0 \oplus b_0)|\cdots|(v_5 \oplus b_5)|(v_6 \oplus 80)|(v_7 \oplus 00)) \end{aligned}$$

Conversely the decryption operation outputs

$$\begin{aligned} \mathsf{ENC}^{-1}(C) \oplus V &= (v_0 \oplus b_0)|\cdots|(v_5 \oplus b_5)|(v_6 \oplus 80)|v_7 \\ &\quad \oplus \\ &\quad v_0|\cdots|v_7 \\ &= b_0|\cdots|b_5|80|00 \\ &= B \end{aligned}$$

Let us consider an adversary who replaces V with $\tilde{V}$. If the block $\tilde{V}$ is randomly generated, likely the decryption operation does not yield a valid padding data. If the block $\tilde{V}$ is carefully chosen by replacing the last two bytes of V $v_5|v_6$ with $(v_5 \oplus g \oplus 80)|(v_6 \oplus 80)$, where g is some

[3] Some encryption modes (e.g., [Nat04]), coupled with a security proof [Jon03], correspond to the MAC-then-Encrypt (MtE) paradigm.

byte, then the decryption yields the following result

$$\begin{aligned}\mathsf{ENC}^{-1}(C) \oplus \tilde{V} &= (v_0 \oplus b_0)|\cdots|(v_5 \oplus b_5)|(v_6 \oplus \mathtt{80})|v_7 \\ &\quad \oplus \\ &\quad v_0|\cdots|v_4|(v_5 \oplus g \oplus \mathtt{80})|(v_6 \oplus \mathtt{80})|v_7 \\ &= b_0|\cdots|b_4|(b_5 \oplus g \oplus \mathtt{80})|\mathtt{00}|\mathtt{00}\end{aligned}$$

The decryption operation is correct if and only if the padding data is valid. In our example, this depends on the value $b_5 \oplus g \oplus \mathtt{80}$:

- If $b_5 \oplus g \oplus \mathtt{80} = \mathtt{80}$, then the padding data is valid.
- If $b_5 \oplus g \oplus \mathtt{80} \neq \mathtt{80}$, then the padding data is (likely) invalid.[4]

Obviously $b_5 \oplus g \oplus \mathtt{80} = \mathtt{80}$ if and only if $b_5 = g$. In other words, the result of the decryption operation (more precisely, the padding verification) reveals the value of the plaintext byte b_5: knowing if the padding data is valid yields b_5.

Iterating that process with different blocks $\tilde{V}$ produced by replacing successively $v_i|v_{i+1}$ with $(v_i \oplus g \oplus \mathtt{80})|(v_{i+1} \oplus b_{i+1})$, where b_{i+1} is the plaintext byte found during the previous step, allows retrieving the plaintext byte b_i. For each byte b_i the attacker wants to retrieve, a new ciphertext $\tilde{V}\|C$ is built with different values g, and each resulting ciphertext is sent to an oracle $\mathcal{O}$, which the attacker needs to get access to. The oracle returns 1 if the padding data is valid, 0 otherwise. The response from the oracle eventually reveals if the attacker's guess g is correct or not. This procedure eventually provides all the $n = 6$ plaintext bytes $b_0, \ldots, b_5$. As we can see, the attack leans on the malleability of the CBC mode, and uses the block V as a pivot in order to get bytes encrypted within C.

The overall attack is described by Algorithm 4.1 (we assume without loss of generality that the targeted block C includes at least one byte of padding, i.e., 80).

If the plaintext to retrieve corresponds to more than two blocks B_0, ..., B_{k-1}, $k > 2$ (which encryption yields the blocks C_0, ..., C_{k-1}), the attacker applies Algorithm 4.1 by using successively each pair of encrypted blocks (C_{k-2}, C_{k-1}), (C_{k-3}, C_{k-2}), ..., (C_0, C_1), (C_{-1}, C_0) where C_{-1} is the CBC IV (equal to $\mathtt{00}^8$ in SCP02). In the remainder of this chapter we assume that the ciphertext is made of two blocks V, C.

Note that the length of the padding is helpful since knowing this value shortens the duration of the attack. Yet it is not necessary since the attack can retrieve the padding data as any other unknown plaintext byte. If unknown, the padding length can be found using the dichotomous algorithm proposed by Black and Urtubia [BU02] (which complexity is $\log_2(d)$ where d is the block size), or a linear search by testing all bytes starting from the rightmost one until the decryption of the modified block yields a valid padding (this method finds the length in $\min(d, h+1)$ steps where h is the padding length).

4.4 Application to SCP02

In this section, we explain how to apply the padding oracle attack presented in Section 4.3 to the particular case of SCP02.

[4] If $b_5 \oplus g \oplus \mathtt{80} = \mathtt{00}$ this may also yield a valid padding (if the preceding decrypted bytes are equal to $\mathtt{80}\ \mathtt{00}^j$, $j \geq 0$). But this (rare) case is easy to manage.

Algorithm 4.1 Regular padding oracle attack

FindBlock(V, C)
  for $i = n - 1$ down to 0
    $b_i \leftarrow$ **FindByte**(b_{i+1}) // $b_n = \texttt{80}$
  end for
  return $b_0 \cdots b_{n-1}$

FindByte(b)
  for $g = \texttt{00}$ to $\texttt{FF}$
    $\tilde{V} \leftarrow$ in V replace $v_i|v_{i+1}$ with $(v_i \oplus g \oplus \texttt{80})|(v_{i+1} \oplus b)$
    send $\tilde{V}\|C$ to the oracle $\mathcal{O}$
    $r \leftarrow \mathcal{O}(\tilde{V}\|C)$
    if $r = 1$
      return g
    end if
  end for

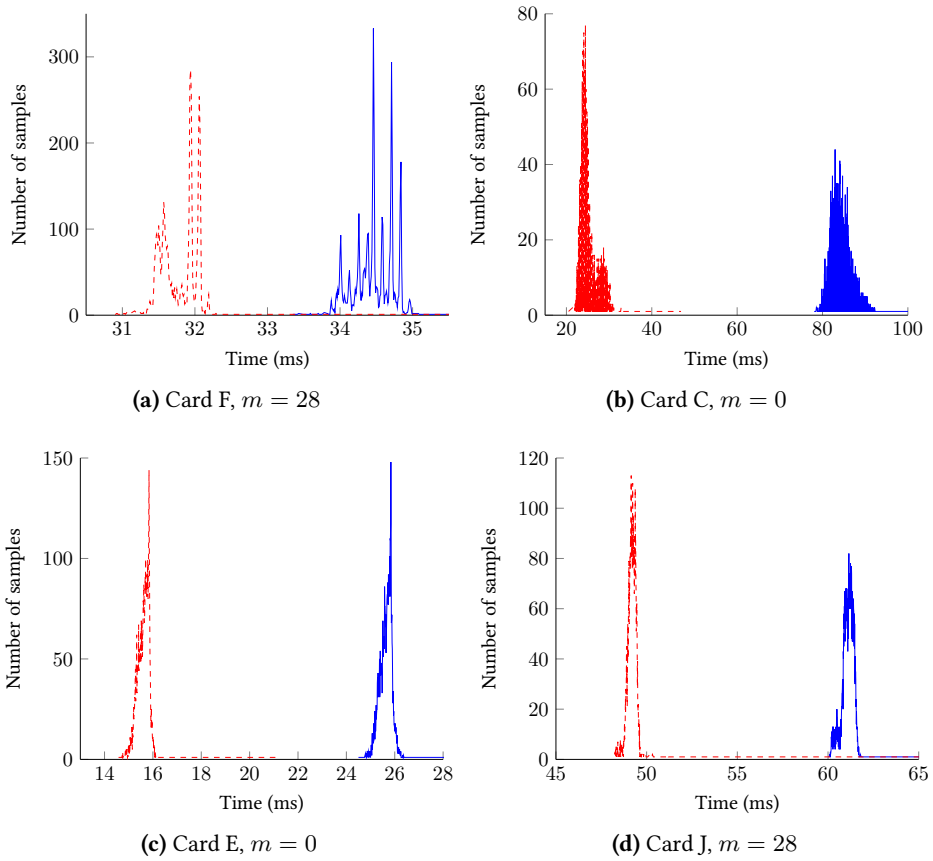
4.4.1 Timing Side-channel

As soon as the valid ciphertext $V\|C$ is changed into $\tilde{V}\|C$, the MAC verification yields an error, the padding data being valid or not, because the ciphertext is modified. In SCP02 this ends with the smart card outputting an error code. Furthermore, among the smart cards we have tested, an error code is *always* sent, and the *same* error code is provided whatever the validity of the padding data. Therefore we have used the time spent by the smart card during the whole decryption procedure to build the oracle $\mathcal{O}$.

The experiments we have done with several cards (labeled Card A to Card J) show that the card's response time when the padding data is valid is higher compared to the case when the padding data is invalid. This leads to two observations. Firstly, the MAC tag is seemingly not verified when the padding data is invalid. Secondly, the response time reflects the computation time. Depending on the smart card, the timing difference ranges from quite low to unexpectedly high. For instance, for Card B, the timing difference when the padding data is valid (15.60 ms) and invalid (14.83 ms) is less than 1 ms, while for Card C, the timing difference between both cases (84.34 ms vs. 25.17 ms) is higher than 59 ms. The figures we provide correspond to the expected values regarding both padding possibilities, and using a command that carries two encrypted blocks and the 8-byte MAC tag.

The experiments we have made with each card show that the two distributions corresponding to the response time when the padding is valid (D_R) and when it is invalid (D_W) are clearly distinguishable and almost disjointed. Figure 4.2 illustrates the results corresponding to four of the attacked smart cards. For each configuration (valid and invalid padding), 5000 encrypted commands have been sent to the smart card, in order to get these measurements.

Let t_{min} be a lower bound of the values corresponding to D_R. If a response time is lower than t_{min} it is highly likely a wrong guess (since we never observed that a correct guess corresponds to a response time lower than t_{min}). Hence, in such a case, one attempt only allows discarding an incorrect value. On the other hand, if a response time is higher than this threshold t_{min}, it is rather likely a correct guess (since we observed only a very few number of values corresponding to D_W that are higher than this threshold). Yet it is also possible that it corresponds to a wrong

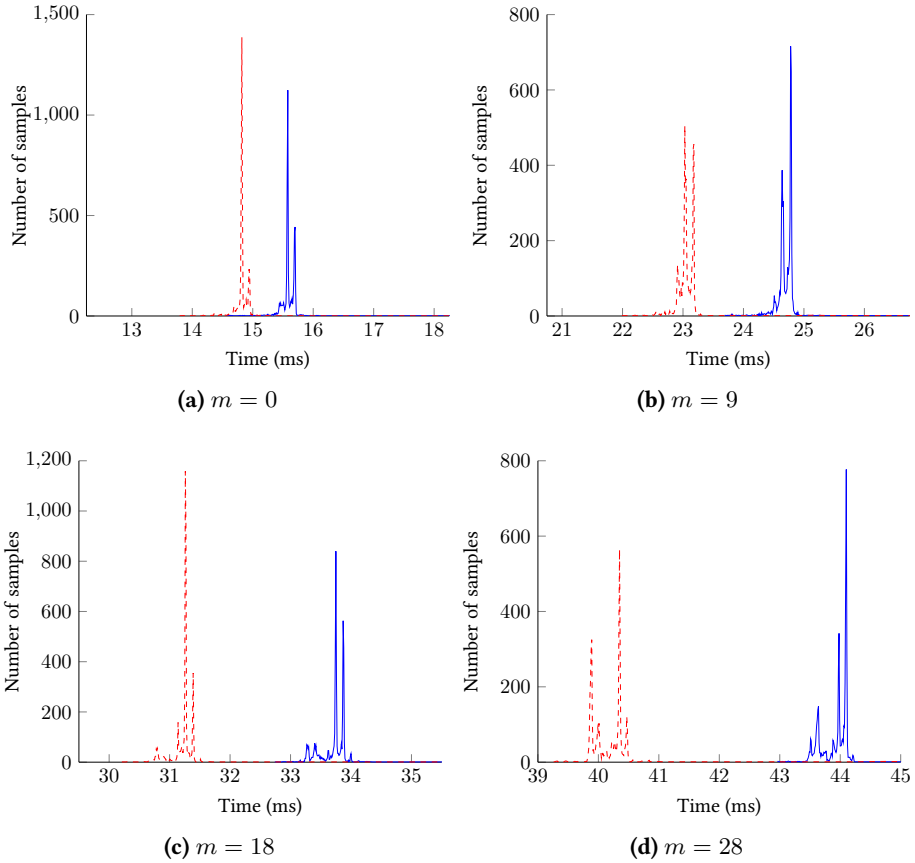


(a) Card F, $m = 28$ **(b)** Card C, $m = 0$

(c) Card E, $m = 0$ **(d)** Card J, $m = 28$

Figure 4.2 – Distribution of the number of cases when the padding data is valid (D_R, continuous blue line) and when it is invalid (D_W, dashed red line) with respect to the response time. The command's data field carries $m+2$ encrypted blocks (including padding data), and the 8-byte MAC tag.

guess with an unexpected long response time. Therefore, in order to detect a right guess, several trials may be necessary. As we will see in Section 4.4.2, this heuristic allows having a success probability close to 1. Moreover, in order to increase the discrepancy of the response time when the padding is valid and when it is invalid, we prepend extra blocks $R_0, \ldots, R_{m-1}$ to the ciphertext, following the same technique as [CHVV03], so that the MAC verification (when it is actually done) involves also these additional blocks. That is, instead of $V\|C$, the smart card receives as the encrypted data the string $R_0\| \cdots \|R_{m-1}\|\tilde{V}\|C$. Indeed, during the decryption operation, the DES function (or its inverse) is called around $3q$ times, where q is the number of plaintext blocks. During the MAC verification, the same function is called at least $q+2$ times.[5] Therefore the timing difference separating a valid and an invalid padding is proportional to roughly $q+2$. As an illustration, Figure 4.3 depicts the distributions D_W and D_R corresponding to Card B for several values m (for each graph, the abscissa axis has the same 6-ms amplitude). The difference between the mean values corresponding to D_R and D_W is 4.6 times higher when $m = 28$ (3.7 ms) compared to $m = 0$ (0.8 ms).



(a) $m = 0$ **(b)** $m = 9$

(c) $m = 18$ **(d)** $m = 28$

Figure 4.3 – D_R (continuous blue line) and D_W (dashed red line) corresponding to Card B with different values m.

[5] Optionally the IV used during the MAC computation is encrypted. In such a case the DES function is called at least $q+3$ times during that operation.

4.4.2 Leveraging the Timing Side-channel

Complexity. We analyse the complexity of the attack applied to SCP02 following the same reasoning as Canvel et al. [CHVV03].

Let K_R be the number of attempts necessary to detect a right guess, and K_W the number of attempts necessary to discard a wrong guess. Let ϵ_+ be the probability of a bad decision when the distribution is actually D_W, and ϵ_- the probability of a bad decision when the distribution is D_R. Let p_i be the probability to find the byte b_i, $0 \leq i < n$. If we assume that all bytes are independent, then the probability to find the n bytes is $p = p_0 \times \ldots \times p_{n-1}$. Moreover if the bytes are uniformly drawn at random from a set of size ℓ, each byte has the same probability to be correctly found. Therefore $p = p_0^n$. Each byte can take any of the ℓ possible values. A guess g is equal to a byte b_i if and only if

- the choice for g is correct (which probability is $\frac{1}{\ell}$),
- incorrect values for g are discarded (which probability is $(1-\epsilon_+)^j$ where j is the number of incorrect values tried before the correct one),
- and the correct value for g is detected (which probability is $1-\epsilon_-$).

Hence

$$\begin{aligned} p_0 &= \sum_{j=0}^{\ell-1} \frac{1}{\ell}(1-\epsilon_+)^j(1-\epsilon_-) \\ &= \frac{1-\epsilon_-}{\ell} \times \frac{1-(1-\epsilon_+)^\ell}{\epsilon_+} \\ &\simeq (1-\epsilon_-)(1-\epsilon_+)^{\frac{\ell-1}{2}} \end{aligned}$$

if $\epsilon_+ \ll \frac{1}{\ell}$. Therefore

$$p \simeq (1-\epsilon_-)^n(1-\epsilon_+)^{n\frac{\ell-1}{2}}$$

Let t be the response time corresponding to a trial with some value g. Let us assume that all trials are independent. Let τ_+ and τ_- be respectively $\Pr[t > t_{min}$, when the distribution is $D_W]$ and $\Pr[t \leq t_{min}$, when the distribution is $D_R]$. Following our heuristic, we discard a guess g as soon as the corresponding response time t is lower than t_{min} (i.e., $K_W = 1$). Therefore the event corresponding to ϵ_+ occurs when the distribution is D_W, if $t > t_{min}$ K_R successive times. Hence

$$\epsilon_+ = \tau_+^{K_R}$$

The event corresponding to ϵ_- occurs when the distribution is D_R, if $t > t_{min}$ at most $K_R - 1$ times, and $t \leq t_{min}$ the subsequent trial. Therefore

$$\epsilon_- = \sum_{j=0}^{K_R-1} (1-\tau_-)^j\tau_- = 1-(1-\tau_-)^{K_R}$$

Hence, we have that

$$p \simeq (1-\tau_-)^{n\cdot K_R}\left(1-\tau_+^{K_R}\right)^{n\frac{\ell-1}{2}}$$

Moreover t_{min} is defined such that $\tau_- = 0$. Therefore, this simplifies into

$$p \simeq \left(1-\tau_+^{K_R}\right)^{n\frac{\ell-1}{2}}$$

If each byte b_i is uniformly drawn at random among ℓ possible values, the average number of values g to be tested before the right one is found is $\frac{\ell+1}{2}$.

The average number of trials to find one byte is $\frac{\ell-1}{2}K_W + K_R$. The overall complexity is then $Z = n \times \left(\frac{\ell-1}{2}K_W + K_R\right)$ to retrieve a n-byte string. Since the burden in the complexity lies on the number of wrong attempts, slightly increasing K_R allows increasing the overall probability of success p while it marginally increases the complexity Z. Indeed, if $\tau_+ < 1$, p is an increasing function of K_R.

Application to SCP02. The attack is then the following. When looking for a byte b_i, for each possible value g, a modified ciphertext $\tilde{V}\|C$ is sent to the targeted smart card. The different response times are collected until a decision can be taken (which means in practice $K_W \simeq 1$ attempt to discard a wrong guess and $K_R \in \{1, 2, 3\}$ attempts to detect a right guess). If the decision is that the guess is correct, the trials are stopped and the attack continues with the byte b_{i-1}. Otherwise, another guess value g is tested.

Algorithm 4.2 describes this timing attack (we assume without loss of generality that C includes at least one byte of padding, i.e., `80`). For a given value g, the **Stop** procedure returns `true` as soon as a response time is lower than t_{min} or if the number of successive response times higher than t_{min} reaches K_R. The **Correct** procedure returns `true` if the number of successive response times higher than t_{min} is equal to K_R.

Algorithm 4.2 Padding oracle attack based on the card response time

FindBlock
  for $i = n - 1$ down to 0
    $b_i \leftarrow$ **FindByte**$(b_{i+1}, \ldots, b_n)$ // $b_n =$ `80`
  end for
  return $b_0 \cdots b_{n-1}$

FindByte$(b_{i+1}, \ldots, b_n)$
  for $g =$ `00` to `FF`
    $j = 0$
    do
      get a new ciphertext $V\|C$
      $\tilde{V} \leftarrow$ in V replace $v_i|v_{i+1}|\cdots|v_n$
        with $(v_i \oplus g \oplus$ `80`$)|(v_{i+1} \oplus b_{i+1})|\cdots|(v_n \oplus b_n)$
      send $R_0\|\cdots\|R_{m-1}\|\tilde{V}\|C$ as encrypted data to the smart card
      $t_j \leftarrow$ response time
      $j = j + 1$
    until Stop$(t_0, \ldots, t_{k-1}) =$ `true`
    if Correct$(t_0, \ldots, t_{k-1}) =$ `true`
      return g
    end if
  end for

As soon as a cryptographic error occurs on the smart card side (e.g., the smart card receives a message with an invalid MAC tag, which happens when the attacker changes $V\|C$ into $\tilde{V}\|C$),

the session is stopped. Therefore a new session has to be started in order to perform each trial. Hence a new ciphertext $V\|C$ (corresponding to the same plaintext that the attack aims at retrieving) must be obtained. Since a new ciphertext is used for each attempt, we have to take into account the bytes $b_{i+1}, \ldots, b_n$ already found, and change V accordingly.

4.4.3 Discussion

The attack we have described is fully successful in our experimental setting. The necessary conditions in order to succeed are the following ones:

1. The attacker sits between the remote server and the card at a point where she can directly eavesdrop on SCP02 encrypted commands and send modified commands to the card.
2. The attacker is able to discriminate response times corresponding to a valid and an invalid padding.
3. The remote server repeatedly sets up a (new) secure channel with the card.
4. The same secret information is sent through each such secure channel.
5. The secret information is sent at a predictable position.

Real case scenario. The attack can be tried in the following real use-case of SCP02: the upload of an applet into an UICC that is plugged into a smartphone. In order to send the applet to the UICC, a SCP02 channel is established between a remote server and the UICC, through the intermediary of an application on the smartphone. In addition, another secure channel can be established between the remote server and the application on the smartphone. That is, the SCP02 commands that carry the applet are embedded into this second secure channel (e.g., TLS or SCP80). Once the data are received by the application on the smartphone, they are decrypted (with respect to TLS or SCP80), and the output (i.e., the encrypted SCP02 commands) is sent to the UICC. Therefore the applet is protected only by the means of SCP02 between the application and the UICC. SCP02 commands (e.g., the `STORE DATA` command) are used to upload such an applet, which may carry secret data (e.g., a symmetric key used to encrypt data, or to authenticate the user with respect to the service provided by the applet). A malicious application or a Trojan on the smartphone can apply the attack in order to break the SCP02 channel, and to retrieve these sensitive data. It behaves as a local man-in-the-middle attacker between the application on the smartphone and the smart card (see Figure 4.4).

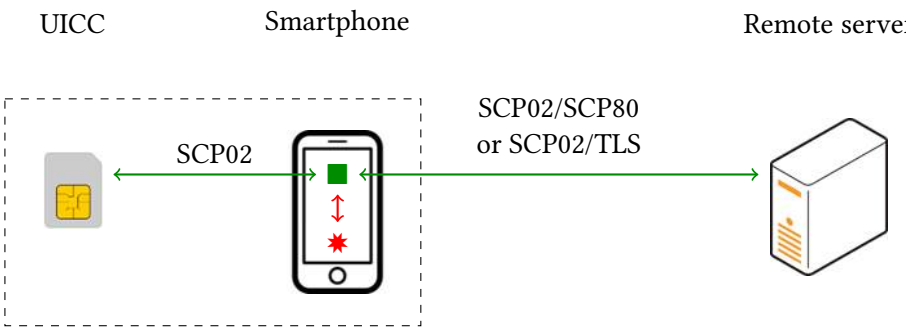


Figure 4.4 – Padding oracle attack targeting an UICC. The malicious application (✸) has access to the memory space of the legitimate application (■) on the victim's smartphone.

The attacker can proceed as follows. First she succeeds in getting the victim download a malicious application (embedded in an apparently inoffensive application deployed on a popular store) into his smartphone. Then the malicious application can use a vulnerability lying in the legitimate application in order to escalate privileges, and to get access to the application memory space as described by Davi, Dmitrienko, Sadeghi, and Winandy [DDSW11]. The latter assumes that a flaw in the legitimate application is exploitable. Alternatively the attacker can use a Trojan embedded in a popular application, or in a game (such as Trojans Dvmap or Tordow in Pokemon Go or Telegram). Once into the smartphone, the Trojan may escalate privileges to gain root access [Kiv], or inject a malicious code into system libraries, which will then be executed with system rights [Unu]. To that point, the Trojan is able to get access to the memory space of the legitimate application. Hence, it can read the genuine SCP02 commands, and modify it. The crafted command is then sent to the UICC, completing the process initiated by the legitimate application.

Fulfilling the conditions of success. Let us assume that encrypting the applet with SCP02 yields k blocks C_0, ..., C_{k-1} (possibly transmitted to the card through several successive `STORE DATA` commands), and that a 16-byte symmetric key lies in the blocks C_i, C_{i+1}, $i+1 \leq k-1$. The number of SCP02 sessions a card can establish is bounded (to 2^{15}) as per specification. Hence the number of trials the attacker can make is also limited. Therefore if (and only if) the attacker needs to decrypt the whole encrypted applet, it may happen that the key be out of reach. However, if the attacker knows the position of the key within the encrypted data (condition 5), she can directly target the corresponding encrypted blocks (whatever the size of the encrypted applet). In such a case, the attacker uses first the blocks C_i, C_{i+1} as described by Algorithm 4.2. The block C_i is changed into $\tilde{C}_i$ in order to incorporate the attacker's guess g, and the string $\tilde{C}_i \| C_{i+1}$ is placed into a SCP02 command data field (with a trailing 8-byte arbitrary MAC tag, and an optional leading string $R_0 \| \cdots \| R_{m-1}$). This fake SCP02 command is then sent to the UICC by the malicious application. Based on the time elapsed until the UICC sends back a response, the attacker knows if her guess is correct or not. The attacker uses the blocks C_i, C_{i+1} to retrieve the 8 bytes of the symmetric key encrypted as C_{i+1}, and the blocks C_{i-1}, C_i in order to get the 8 bytes of the symmetric key encrypted as C_i.

The attacker expects a clean and stable response time (condition 2). Yet it may happen that this time value be influenced by surrounding activities. In such a case more sophisticated statistical tools may be used by the attacker (e.g., see [CHVV03; AP16]).

Another necessary condition is that the same secret be repeatedly sent through the secure channel (condition 4). The nominal behaviour of the underlying application protocol could not be as expected by the attacker. Yet this condition could be fulfilled if, upon reception of the cryptographically invalid command (modified by the attacker), the underlying application within the card triggers a message asking the server for a new delivery of the same message (which contains the secret). Also, upon reception of an error code, the server could decide to send again the command carrying the secret. If the remote server sends again this command only or the whole applet, it is very likely that the symmetric key lie at the same position within the command and the applet. Hence condition 5 remains fulfilled.

This error code may be sent either by the card itself (because the command it received has been modified by the attacker), or by the attacker. By default, in SCP02, the responses sent by the card are not encrypted nor protected with a MAC tag. Then the server may keep sending the same command (if not everlastingly, at least many times, which allows the attacker to retrieve a substring of the secret) until it is acknowledged by the card.

The complexity per byte is $\frac{\ell-1}{2} K_W + K_R$. The figures provided in Section 4.4.4 assume that

each byte is uniformly distributed over $\{\texttt{00}, \ldots, \texttt{FF}\}$. However the alphabet size the secret bytes belong to can be lower than $\ell = 256$ (e.g., the secret is some sort of PIN code or password). That would decrease the overall complexity. Moreover it may also be possible for the attacker to retrieve only a substring of the secret and then complete the attack through an exhaustive search or a dictionary attack, if the attacker is able to test the remaining values.

Furthermore, the attack is made easier if the targeted card does not close the secure channel when it receives an invalid command (the one modified by the attacker). In such a case the attacker needs to eavesdrop on the encrypted command (that carries the secret data) once only. Then she can reuse the same encrypted version of the secret to perform the successive changes (with the different values g), and send the modified commands to the card. This would remove the conditions 3 and 4 listed above, necessary to the attack. Yet a card behaving so would diverge from the SCP02 specification. We stress that none of the smart cards we have tested behave so.

4.4.4 Experimental Results of the Complete Attack

Test bench. We have developed the attack with the Java OPAL library [SSD] which implements several protocols for smart cards (including SCP02). The communication with the smart card and the reader is managed with the `javax.smartcardio` package. The experiments have been done on two laptops HP EliteBook and Dell Latitude E6430 running on Windows 7. Four card readers have been used: the inner laptop readers (Broadcom Corp Contacted SmartCard, Alcor Micro USB Smart Card Reader), a contact reader (Omnikey 5321 Smart Card Reader USB), and a contactless one (SpringCard Prox'N'Roll).

Our program simulates both the legitimate server and the attacker. The kinematic is the following. First an n-byte string is pseudo-randomly generated (the secret value the attacker has to retrieve). Then the server procedure sets up an SCP02 channel with the smart card and computes shared session keys. Through the channel, an encrypted command carrying the n-byte secret is sent to the card. A copy of that encrypted command is given to the attacker procedure, which modifies it, sends it to the card and measures the response time. The measured time is equal to the elapsed time between the moment the command is sent and the moment a response is received from the smart card (a single Java procedure, `transmit`, handles the command to send and the response to receive). Since the channel is closed by the smart card (because the modified command sent by the attacker procedure is cryptographically invalid), the server procedure sets up a new secure channel and sends again the same n-byte secret. This new encrypted version of the secret is given to the attacker procedure, and so forth.

We have made experiments with 10 smart cards produced by six different card manufacturers (see Table 4.1). Prior to the attack, we have made trials in order to get the distributions D_W and D_R for each smart card. This is straightforward since we own the card keys. D_R corresponds to a valid padding (and a wrong MAC tag), and D_W corresponds to an invalid padding (and a wrong MAC as well). We were able to use an invalid padding during the encryption procedure. Yet in a real context, the adversary that has access to an SCP02 encryption oracle is able to get a distribution very close to D_W by changing the trailing bytes of the encrypted data. With high probability, changing these encrypted bytes yields an invalid padding during the decryption procedure. Alternatively, the attacker can perform offline tests with a specimen of the targeted card.

Experiments and results. As the smart card was the target, we have used commands and not responses to launch the attack. We have used different types of SCP02 command (`PUT KEY`,

STORE DATA, GET DATA, GET STATUS), including commands that are not supposed to carry an application payload (however these commands, when protected, carry the encrypted padding and the MAC tag). Yet our purpose is to show that it is indeed possible to retrieve plaintext bytes whatever the command used, be it a command not supposed to carry data besides possible flags (e.g., GET STATUS, GET DATA), or commands which are intended to transmit sensitive data to the smart card (e.g., PUT KEY, STORE DATA).

We observe that the payload carried in a PUT KEY command is encrypted with 3DES in CBC mode, as any command payload in secure mode. However, prior to be transmitted through the secure channel, the plaintext data is first encrypted (with 3DES and another session key than the one used to provide data confidentiality throughout the secure channel). We stress that we do not break this inner encryption layer. Yet, the padding oracle attack allows retrieving all the (encrypted) bytes sent through the secure channel even when this "sensitive" command is used.

The maximum data size for a command is 255 bytes. Without the MAC tag, there remain 247 bytes $= 30$ DES blocks $+ 7$ bytes. The attack involves two useful blocks V and C. This leaves at most 28 extra blocks R_i in order to amplify the timing discrepancy. In practice we have used either 0 or 28 random blocks depending on the targeted smart card.

In the best case (i.e., $K_W = K_R = 1$), the average complexity to retrieve n bytes of plaintext is $Z = n \times \frac{\ell+1}{2}$. The number of available SCP02 sessions is at most 2^{15} (i.e., the maximum value of a keyset's sequence counter). Since a new session has to be established for each trial, the maximum number of plaintext bytes that can be retrieved in the best case is bounded by $2 \times \frac{2^{15}}{\ell+1} < 2^8$, if $\ell = 256$.

Table 4.1 – Experimental results for the padding oracle attack with $n = 16$ (pseudo-random) bytes of plaintext.

Manufacturer[a]	Card	μ_W (ms)	μ_R (ms)	t_{min} (ms)	m	τ_+ (%)	K_W	K_R	Z	Z/n
1	A	39.60	42.59	41.00	28	0.16	1	3	2055.71	128.48
	B	40.19	43.94	42.00	28	0.44	1	3	2077.78	129.86
2	C	25.17	84.34	75.00	0	0.00	1	2	2043.95	127.75
	D	26.64	34.36	32.00	0	0.00	1	2	2066.54	129.16
3	E	15.61	25.65	23.00	0	0.00	1	2	2134.03	133.38
4	F	31.81	34.48	33.00	28	0.48	1	3	2109.71	131.86
	G	15.64	18.53	17.00	0	0.28	1	3	2103.62	131.48
5	H	25.18	84.86	72.00	0	0.00	1	2	2048.34	128.02
6	I	25.90	35.85	32.00	0	0.06	1	3	2108.60	131.79
	J	14.32	19.92	17.50	0	0.10	1	2	2094.85	130.93

[a]For confidentiality purpose, the manufacturer names and the card identifiers are not provided in this thesis.

Table 4.1 summarises our results. The figures provided are the mean corresponding to the 300 tests roughly that we have performed with each smart card. For each card, we provide the expected response time in case of an invalid padding (μ_W) and valid padding (μ_R), the threshold t_{min} used to detect a correct guess, the number m of additional blocks R_i used to increase the computation time discrepancy, the probability τ_+ that an invalid padding yields a high response

time, the number of trials to discard a wrong attempt (K_W), the number of trials to detect a right attempt (K_R), the average complexity Z to retrieve $n = 16$ bytes, and the average complexity per byte (Z/n).

In our experiments, the minimum value for K_R is 2 in order to be able to correctly find a plaintext byte equal to 80 (in such a case, at least one additional attempt with the same value g is made). But in fact, for some cards (Card C, Card D, Card H), $K_R = 1$ is enough to detect a right guess.

As we can see, the complexity per byte Z/n is almost optimal (close to 128.5). Moreover the heuristic we use is actually valid since the probability τ_+ that a wrong guess yields a high response time (i.e., above t_{min}) is low.

The duration of the attack to retrieve $n = 16$ bytes ranges roughly from 160 s (Card A, Card B) up to 680 s (Card C). It depends mainly on the smart card itself (some card sends a response faster than others).

Estimating the number of smart cards affected by this vulnerability is not easy. However cards likely implementing SCP02 are produced in their billions every year [SIM18; GSM19]. Therefore, the number of impacted smart cards is potentially high.

4.5 Countermeasures

MAC the padding data. Several countermeasures aiming at mitigating a padding oracle attack have been proposed formerly. A simple fix proposed by Vaudenay [Vau02] is to pad the plaintext first, and then to compute the MAC tag on the padded data. During the whole decryption procedure, the MAC tag must be verified first, and the padding data removed after.

Resistant padding scheme. As observed by Black and Urtubia [BU02], and proved by Paterson and Watson [PW08], slightly changing a byte-oriented padding of the form $\texttt{80}\ \texttt{00}^i$ into a bit-oriented padding of the form $\texttt{10}\cdots\texttt{0}$ makes the padding oracle (almost) vanish. Indeed, any decrypted block is correctly padded with respect to a padding of the latter form unless the decrypted block contains no bit equal to 1 (i.e., the block is equal to $\texttt{00}^d$, where d is the block byte length). Therefore the padding oracle is not usable any more if each byte of the plaintext block is uniformly drawn at random, because the attacker cannot look for each byte independently but has to enumerate all possible block values. Yet, as noticed by Black and Urtubia, a dictionary attack may still be conceivable if the attacker looks for a secret value belonging to a reduced size set (if the attacker guesses correctly, the decryption yields $\texttt{00}^d$, which is the only invalid plaintext). This bit-oriented padding scheme is recommended for use with CBC mode by ISO [Int17], with a slight difference: the number of optional 0 bits must be as few as possible. Therefore the padding data is carried in one block at most. This means that any plaintext which last block is all-zero is invalid with respect to this padding scheme.

Another scheme resisting padding oracle attacks and proposed by Black and Urtubia is the following. An arbitrary byte x distinct from the last plaintext byte is picked, and the data is padded with x. The receiver removes all matching trailing bytes until either a distinct byte is found or the empty string is reached. With respect to this padding scheme, any decrypted block is valid. Therefore it seems that the oracle is removed. We observe however that even this scheme may still provide an oracle, depending on the behaviour of the receiver when the decryption outputs an empty string. If this yields a distinct error or a discrepancy in the calculation duration, that could be exploited in order to perform a (dictionary) attack. Let $V\|C$ be the encryption of some secret value B. The attacker tries all possible values B' and changes V into $V' = V \oplus B'$. Then the decryption of $V'\|C$ yields $\mathsf{ENC}^{-1}(C) \oplus V' = (V \oplus B) \oplus (V \oplus B') = B \oplus B'$. If $B \oplus B'$

contains at least two distinct bytes, then there is at least one remaining byte after the padding extraction. If all bytes in $B \oplus B'$ are equal, then no byte remains after the padding removal. If the attacker is able to detect such a case, she knows that $B = B' \oplus (y| \cdots |y)$ where each byte y can take at most 256 values. Therefore this leaves at most 256 candidates for B.

Equal computation or response time. Canvel et al. [CHVV03] suggest to make error responses time-invariant by simulating a MAC verification even when there is a padding error. This implies carefully implementing the decryption procedure in order to eliminate all timing channels. Indeed AlFardan et al. [AP13] have shown that even a tight channel can be exploited. In addition, the latter authors warn that adding random delays during the decryption procedure may not be sufficient if the delays follow a uniform distribution. This change would merely increase the complexity of the attack.

In turn, Askarov, Zhang, and Myers [AZM10] propose a mitigation scheme that applies to a broad class of computations. With respect to SCP02, this means sending the responses at scheduled times (that is somehow padding the response time to an upper bound). These authors observe that this does not prevent timing leaks, yet it bounds the amount of information provided through this side channel as a function of the elapsed time.

Resistant operation mode. Another fix is to replace the CBC mode with an authenticated encryption algorithm as suggested by Kupser, Mainka, Schwenk, and Somorovsky [KMSS15].

The latter proposal or another construction than E&M may be a suitable choice since Paterson and Watson [PW12] extend the results of Bellare and Namprempre [BN08] to prove that the E&M construction using CBC mode with a padding method as its encryption component is not generically secure in the chosen ciphertext setting.[6] They also prove that the EtM construction (known to be secure in general when padding is not considered) is also secure when a padding method is used.

`PUT KEY` command. In addition to these countermeasures, in the SCP02 case, one may use the `PUT KEY` command in order to send secret values to the smart card (e.g., symmetric keys). The data field of such a command corresponds to the output of a double encryption process: first with a session key different than the one used to encrypt and MAC the other commands, then with these same secure channel session keys. Therefore, applied to a `PUT KEY` command, the attack breaks the upper encryption layer but not the inner one, and yields the data encrypted under this additional session key but not the genuine plaintext data.

Server side. Furthermore, the attack can be mitigated by limiting, on the server side, the number of times the same secret value is sent to the smart card. Since the attacker needs to make several trials (each one implying setting up a new secure channel) in order to get one byte of plaintext, this simple mitigation forbids or drastically reduces the amount of data the attacker can retrieve.

[6] Bellare, Kohno, and Namprempre [BKN04] prove that a particular Encode-then-Encrypt-and-MAC construction is secure when a padding method is used. But the encoding function is assumed to be collision resistant (the computation of the authentication tag involves a sequence number unique per message that aims at precluding collisions between two encoded messages).

5 Security Models

The provable security approach (or, more adequately, the reductionist security approach) is a paradigm aiming at providing convincing evidences regarding the security of a cryptographic scheme. This approach can be used by means of an adversarial model where the adversary faces a challenger and tries to break some security property, during a specific experiment where rules are defined and powers attributed to the adversary.

In this chapter, we use this approach and define two security models. The first model, that we call 3-AKE, aims at analysing the security of 3-party authenticated key exchange protocols (AKE). This model captures in particular the forward secrecy property. In Chapter 7, we present a generic 3-party AKE and, based on it, a concrete instantiation, in the symmetric-key setting. We use this model to formally prove the security of these two schemes.

The second security model is based on the ACCE paradigm. It allows analysing 3-party protocols whose goal is to setup secure channels, hence its name 3-ACCE. First, we use this framework to prove the security of a generic LoRaWAN-like protocol. Then we present an adapted version of LoRaWAN 1.1 with stronger security properties, modified in order to mitigate the vulnerabilities described in Chapter 3. Applying the first result, we formally prove the security of this protocol in our model, and describe how to concretely instantiate it.

The results of this chapter have been published in [ACF19] and [CF19].

Contents

5.1 The Need for a Suitable Security Model

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 [BR94] 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., [BWJM97; BPR00; CK01; CBH05; LLM07; MSW08; Cre09; JKSS11] to name a few). Of particular interest are the security models proposed to analyse real-world protocols such as TLS [MSW08; JKSS11; KPW13], IPsec and IKE [FS99; CK02; Cre11], SSH [BKN02]. 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 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. This is illustrated by the works summarised below, and also by the protocol LoRaWAN 1.1.

Existing 3-party security models. Alt, Fouque, Macario-Rat, Onete, and Richard [AFM+16] 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 [FOR16] 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 [BBF+17] consider the use of TLS 1.2 [DR08] 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 *authenticated and confidential channel establishment* (ACCE) model of Jager, Kohlar, Schäge, and Schwenk [JKSS11] to the 3-party setting. It adds in particular to the properties of entity authentication and channel security a third property aiming at "binding" several entities involved in the protocol. Bhargavan et al. 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 [NSV+15] 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 [NLG+17] 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. [NSV+15] and Naylor et al. [NLG+17] 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 these security goals.

In the same context of proxied TLS connections, Bhargavan, Boureanu, Delignat-Lavaud, Fouque, and Onete [BBD+18] 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 [Res18]. 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.

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.

In Section 5.2, we present a security model that allows analysing 3-party authenticated key exchange protocols. This model is used to formally prove the security of the 3-party AKE presented in Chapter 7.

In Section 5.3, we present a security model that allows analysing 3-party LoRaWAN-like protocols, which main goal is to establish secure channels between several parties. This model is motivated by version 1.1 of LoRaWAN, and the vulnerabilities, described in Chapter 3, related to that version.

5.2 The 3-AKE Security Model

In this section, we present our 3-AKE security model. In a nutshell, we use the security experiments of a 2-AKE model (entity authentication, key indistinguishability), as described by Brzuska, Jacobsen, and Stebila [BJS16] (see Chapter 2, Section 2.3.1). Taking inspiration from the 3(S)ACCE model of Bhargavan, Boureanu, Fouque, Onete, and Richard [BBF+17], we extend this 2-AKE model to incorporate the three parties of our 3-AKE protocol, and their interleaved operations.

This 3-AKE model aims at analysing 3-party protocols whose main goal is to yield a session key. This contrasts with the model of Bhargavan et al. which is built on the ACCE notion, and consequently is not meant to prove the security of protocols based on the indistinguishabiliy of the session keys. Hence the relevance of our 3-AKE model.

In our 3-AKE security model, the adversary has full control over the communication network. It can forward, alter, drop any message exchanged by honest parties, or insert new messages. Our 3-AKE model captures also forward secrecy.

5.2.1 Execution Environment

Protocol entities. Our model considers three sets of parties: a set $\mathcal{K}$ of Authentication and Key Servers (KS), a set $\mathcal{E}$ of end-devices (ED), and a set $\mathcal{X}$ of servers (XS) that will eventually share session keys with ED. Each party is given a long-term key ltk.

Session instances. Each party P_i maintains a set of instances $P_i.\mathsf{Instances} = \{\pi_i^0, \pi_i^1, \ldots\}$ modeling several (sequential or parallel) executions of the 3-party protocol Π. Each instance

π_i^n has access to the long-term key $P_i.\mathsf{ltk}$ of its party parent P_i. Moreover, each instance π_i^n maintains the following internal state:

- The *instance parent* $\pi_i^n.\mathsf{parent} \in \mathcal{K} \cup \mathcal{E} \cup \mathcal{X}$ indicating the party owning that instance.
- The *partner-party* $\pi_i^n.\mathsf{pid} \in \mathcal{K} \cup \mathcal{E} \cup \mathcal{X}$ indicating the intended party partner. A party in one of the three sets can be partnered with a party belonging to any of the two other sets.
- The *role* $\pi_i^n.\rho \in \{\mathsf{ed}, \mathsf{ks}, \mathsf{xs}\}$ of $P_i = \pi_i^n.\mathsf{parent}$. If $P_i \in \mathcal{E}$, then $\pi_i^n.\rho = \mathsf{ed}$. If $P_i \in \mathcal{K}$, then $\pi_i^n.\rho = \mathsf{ks}$. If $P_i \in \mathcal{X}$, then $\pi_i^n.\rho = \mathsf{xs}$.
- The *session identifier* $\pi_i^n.\mathsf{sid}$ of an instance.
- The *acceptance flag* $\pi_i^n.\alpha \in \{\perp, \texttt{running}, \texttt{accepted}, \texttt{rejected}\}$ originally set to `running` when the session is ongoing, and set to `accepted`/`rejected` when the party accepts/rejects the partner's authentication.
- The *session key* $\pi_i^n.\mathsf{ck}$ set to $\perp$ at the beginning of the session, and set to a non-null bitstring once π_i^n computes the session key.
- The *key material* $\pi_i^n.\mathsf{km}$. If $\pi_i^n.\mathsf{parent} \in \mathcal{K}$ (resp. $\pi_i^n.\mathsf{parent} \in \mathcal{X}$) and $\pi_i^n.\mathsf{pid} \in \mathcal{X}$ (resp. $\pi_i^n.\mathsf{pid} \in \mathcal{K}$), then $\pi_i^n.\mathsf{km}$ 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. Otherwise $\pi_i^n.\mathsf{km}$ is always set to $\perp$.
- The *status* $\pi_i^n.\kappa \in \{\perp, \texttt{revealed}\}$ of the session key $\pi_i^n.\mathsf{ck}$.
- The *transcript* $\pi_i^n.\mathsf{trscrpt}$ of the messages sent and received by π_i^n.
- The *security bit* $\pi_i^n.\mathsf{b} \in \{0, 1\}$ 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).
- The *partner-parties set* $\pi_i^n.\mathsf{PSet}$ stores the *parties* parent of the instances in $\pi_i^n.\mathsf{ISet}$ (including $\pi_i^n.\mathsf{parent}$).

Adversarial queries. The adversary $\mathcal{A}$ is assumed to control the network, and interacts with the instances by issuing to them the queries described below.

In our 3-AKE model, we use familiar queries. Nonetheless we require some restrictions regarding the Test-query. This query aims at "evaluating" the quality of a key output by any of the 2-AKE runs done during a 3-AKE session. We use the vanilla real-or-random experiment. Nonetheless, some session keys output during a 3-AKE session are used in the same session, which allows the adversary to trivially distinguish between a "real" session key and a random key. Consequently, we forbid the adversary from issuing a Test-query with respect to a key as soon as this key is used (i.e., as input to a function).

Moreover, we require the adversary to be stateless with respect to the Test-query. That is, the key k_b sent in response to a Test-query cannot be used to interact with instances, nor contribute to answering other Test-challenges.

Instead of proving that the key is *good*, one could consider proving that the key is *good to be used for* some purpose [BFS+13; Kra16]. But we chose not to use a weaker notion than the more established ones despite the necessity of these restrictions.

- $\mathsf{NewSession}(P_i, \rho, \mathsf{pid})$: this query creates a new instance π_i^n at party P_i, having role ρ, and intended partner pid.
- $\mathsf{Send}(\pi_i^n, M)$: this query allows the adversary to send any message M to π_i^n. If $\pi_i^n.\alpha \neq$ `running`, it returns $\perp$. Otherwise π_i^n responds according to the protocol specification.
- $\mathsf{Corrupt}(P_i)$: this query returns the long-term key $P_i.\mathsf{ltk}$ of P_i. If $\mathsf{Corrupt}(P_i)$ is the ν-th query issued by the adversary, then we say that P_i is *ν-corrupted*. For a party that has not been corrupted, we define $\nu = +\infty$.
- $\mathsf{Reveal}(\pi_i^n)$: this query returns the session key $\pi_i^n.\mathsf{ck}$, and $\pi_i^n.\kappa$ is set to `revealed`. If $\mathsf{Reveal}(\pi_i^n)$ is the ν-th query issued by the adversary, then we say that π_i^n is *ν-revealed*. For a party that has not been revealed, we define $\nu = +\infty$.
- $\mathsf{Test}(\pi_i^n)$: this query may be asked only once per pairwise partnered instances throughout the game. If $\pi_i^n.\alpha \neq$ `accepted`, then it returns $\perp$. Otherwise it samples an independent key $k_0 \xleftarrow{\$} \mathcal{KEY}$, and returns k_b, where $k_1 = \pi_i^n.\mathsf{ck}$. The key k_b is called the Test-*challenge*. In order to preclude trivial attacks, we forbid the adversary from issuing a Test-query, and answering a Test-challenge as soon as the corresponding key $\pi_i^n.\mathsf{ck}$ is used during the session. Moreover, the adversary is stateless with respect to this query (it does not keep track of k_b).

5.2.2 Security Definitions

Partnership. In order to define the partnership between two instances involved in a 2-AKE run, we use the notion of matching conversations (see Definition 2.1). Consequently, we define sid to be the transcript, in chronological order, of all the (valid) messages sent and received by an instance during a 2-AKE run, 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-AKE 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.

Definition 5.1 (Correctness)**.** *We define the* correctness *of a 3-AKE protocol as follows. We demand that, for any instance π ending in an accepting state, the following conditions hold:*

- $|\pi.\mathsf{ISet}| = 6$. *Let* $\pi.\mathsf{ISet}$ *be* $\{\pi_i^n, \pi_i^m, \pi_k^\ell, \pi_k^s, \pi_j^u, \pi_j^v\}$.
- $\pi_i^n.\mathsf{parent} = \pi_i^m.\mathsf{parent} = P_i \in \mathcal{E}$
- $\pi_k^\ell.\mathsf{parent} = \pi_k^s.\mathsf{parent} = P_k \in \mathcal{K}$
- $\pi_j^u.\mathsf{parent} = \pi_j^v.\mathsf{parent} = P_j \in \mathcal{X}$
- $\pi.\mathsf{PSet} = \{P_i, P_j, P_k\}$
- $\pi_i^m.\mathsf{sid} = \pi_k^\ell.\mathsf{sid} \neq \perp$ *and* $\pi_i^m.\mathsf{ck} = \pi_k^\ell.\mathsf{ck} \neq \perp$
- $\pi_i^n.\mathsf{sid} = \pi_j^u.\mathsf{sid} \neq \perp$ *and* $\pi_i^n.\mathsf{ck} = \pi_j^u.\mathsf{ck} \neq \perp$
- $\pi_k^s.\mathsf{sid} = \pi_j^v.\mathsf{sid} \neq \perp$ *and* $\pi_k^s.\mathsf{ck} = \pi_j^v.\mathsf{ck} \neq \perp$
- $\pi_j^v.\mathsf{km} = \pi_k^s.\mathsf{km} = \pi_i^m.\mathsf{ck} = \pi_k^\ell.\mathsf{ck}$
- $\exists\, g \mid g(\pi_i^m.\mathsf{ck}, \pi_i^n.\mathsf{trscrpt}) = \pi_i^n.\mathsf{ck} = \pi_j^u.\mathsf{ck} = g(\pi_j^v.\mathsf{km}, \pi_j^u.\mathsf{trscrpt})$

The last two conditions aim at "binding" the six instances involved in a 3-AKE run. Function g corresponds typically to the session key derivation function used by P_i (ED) and P_j (XS) together. Figure 5.1 depicts the links between the six instances involved in a correct protocol run.

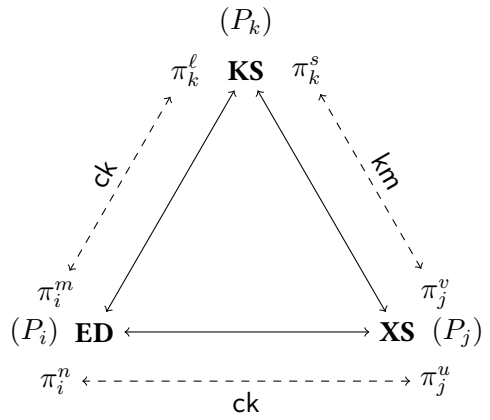


Figure 5.1 – The six instances involved in a correct 3-AKE run

Security of a 3-AKE protocol is defined in terms of an experiment played between a challenger and an adversary. This experiment uses the execution environment described in Section 5.2.1. The adversary can win the 3-AKE experiment in one of two ways: (i) by making an instance accept maliciously, or (ii) by guessing the secret bit of the Test-instance. In both, the adversary can query all oracles NewSession, Send, Reveal, Corrupt, and Test.

Entity authentication (EA). This security property must guarantee that (i) any instance $\pi \in \{\pi_i^n, \pi_i^m, \pi_k^\ell, \pi_k^s, \pi_j^u, \pi_j^v\}$ ending in accepting state is pairwise partnered with a *unique* instance, and (ii) the output of a 2-AKE run done between P_k and P_i is *used as root key* in a 2-AKE run done between P_i and P_j.

Definition 5.2 (EA). *An instance π of a protocol Π is said to* maliciously accept *in the 3-AKE security experiment with intended partner $\tilde{P}$, if*

(a) $\pi.\alpha = \texttt{accepted}$ *and* $\pi.\mathsf{pid} = \tilde{P}$ *when $\mathcal{A}$ issues its ν_0-th query.*

(b) Any party in $\pi.\mathsf{PSet}$ *is ν-corrupted with $\nu > \nu_0$.*

(c) Any instance in $\pi.\mathsf{ISet}$ *is ν'-revealed with $\nu' > \nu_0$.*

(d) There is no unique instance $\tilde{\pi}$ such that $\pi.\mathsf{sid} = \tilde{\pi}.\mathsf{sid}$,
or there is no instances $\pi_i^m, \pi_i^n, \pi_j^u, \pi_j^v \in \pi.\mathsf{ISet}$ *such that*

- $\pi_i^m.\mathsf{pid} = \pi_j^v.\mathsf{pid} \in \mathcal{K}$,
- $\pi_i^n.\mathsf{parent} = \pi_i^m.\mathsf{parent} \in \mathcal{E}$,
- $\pi_j^u.\mathsf{parent} = \pi_j^v.\mathsf{parent} \in \mathcal{X}$, *and*
- $g(\pi_i^m.\mathsf{ck}, \pi_i^n.\mathsf{trscrpt}) = \pi_i^n.\mathsf{ck} = \pi_j^u.\mathsf{ck} = g(\pi_j^v.\mathsf{km}, \pi_j^u.\mathsf{trscrpt})$.

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

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

Key indistinguishability. This security property must guarantee that the adversary can do no more than *guessing* in order to distinguish from random the session key output by any of the 2-AKE runs performed during a 3-AKE protocol session.

Definition 5.3 (Key Indistinguishability)**.** *An adversary $\mathcal{A}$ against a protocol Π, that issues its* Test*-query to instance π during the 3-AKE security experiment,* answers the Test-challenge correctly *if it terminates with output b', such that*

(a) $\pi.\alpha = \texttt{accepted}$

(b) Let $\tilde{\pi}$ be the last instance in $\pi.\mathsf{ISet}$ to end in accepting state: $\tilde{\pi}.\alpha = \texttt{accepted}$ when $\mathcal{A}$ issues its ν_0-th query.

(c) Any party in $\pi.\mathsf{PSet}$ is ν-corrupted with $\nu > \nu_0$.

(d) No instance in $\pi.\mathsf{ISet}$ has been queried in Reveal *queries.*

(e) $\pi.\mathsf{b} = b'$

The adversary's advantage is defined as

$$\mathsf{adv}_{\Pi}^{\text{key-ind}}(\mathcal{A}) = \left| \Pr[\pi.\mathsf{b} = b'] - \frac{1}{2} \right|.$$

The definitions of entity authentication and key indistinguishability allow an adversary to corrupt a party involved in the 3-AKE security experiment (up to some point, in order to preclude trivial attacks). Therefore, protocols secure with respect to Definition 5.4 below provide forward secrecy.

Definition 5.4 (3-AKE Security)**.** *A protocol Π is 3-AKE-secure if Π satisfies correctness, and for all probabilistic polynomial time adversary $\mathcal{A}$, $\mathsf{adv}_{\Pi}^{\text{ent-auth}}(\mathcal{A})$ and $\mathsf{adv}_{\Pi}^{\text{key-ind}}(\mathcal{A})$ are a negligible function of the security parameter.*

5.3 The 3-ACCE Security Model

5.3.1 An Attack Scenario against LoRaWAN 1.1

We have described in Chapter 3, the protocol LoRaWAN 1.1 (Section 3.5) and several flaws impairing the security of the protocol (Section 3.6). Seemingly, the LoRaWAN 1.1 specification considers the protocol as involving two parties only (ED and the back-end network), and does not make clear the security properties it is supposed to guarantee, nor the powers of the adversary it aims at defending against. A third party (namely JS) reduces the security of a client-server type connection (as in LoRaWAN 1.0) by increasing the attack surface. In version 1.1, 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.

We illustrate the latter with the following scenario. 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 attack scenario (see Figure 5.2). 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).

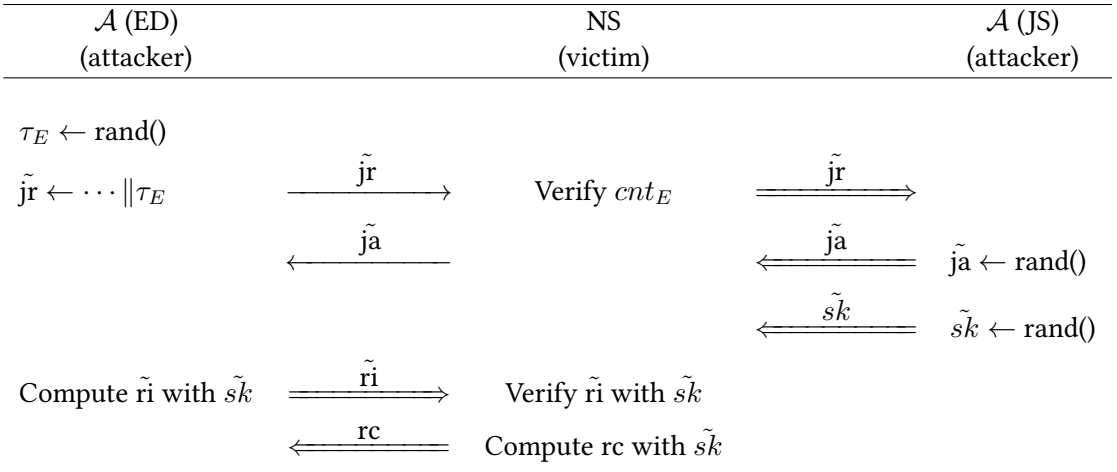


Figure 5.2 – Impersonation of ED to NS based on a weak protocol between NS and JS. Double line arrows indicate the use of secure channel keys.

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. Hence, this attack allows the attacker to impersonate ED to NS although the attacker does not know the ED's master keys, nor does she (directly) target the LoRaWAN 1.1 protocol. It is conceivable because of the way the cryptographic 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 taking 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 Sections 5.3.2 and 5.3.3, we describe a security model, that we call 3-ACCE, which aims at capturing the security goals of such 3-party protocols. In Section 5.3.4 we compare our 3-ACCE model with existing ones, and enlighten the need for such a model. In Section 5.4.1, we use this model to prove the security of a generic LoRaWAN-like protocol. Then, in Section 5.4.2, we provide a security proof of a concrete LoRaWAN 1.1 protocol modified in order to mitigate the flaws presented in Chapter 3.

5.3.2 Execution Environment

We describe the execution environment related to our model, using the notation of the ACCE model of Jager et al. [JKSS11] (see Chapter 2, Section 2.3.2), and Bhargavan et al. [BBF+17].

We use the ACCE paradigm because LoRaWAN 1.1 mixes up the key exchange phase and the post-accept phase. The key exchange phase involves two messages (the so-called RekeyInd and

RekeyConf messages) computed with the session keys, as any other post-accept messages (i.e., sent through the subsequent secure channel). Akin to the Finished messages in TLS 1.2 [DR08], these LoRaWAN messages are used to conclude the key exchange phase. Therefore, we cannot prove the security of the protocol based on indistinguishability of keys, as in an AKE model.

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* $\pi_i^n.\alpha \in \{\perp, \texttt{running}, \texttt{accepted}, \texttt{rejected}\}$ originally set to $\perp$ when the session is ongoing, and set to `accepted`/`rejected` 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 $\pi.\mathsf{PSet} = \{P_i, P_j, P_k\}$, $\forall \pi \in \{\pi_i^n, \pi_j^u, \pi_j^v, \pi_k^\ell\}$.

$\underline{\mathsf{Encrypt}(\pi_i^n, M_0, M_1, H)}$
if $\pi_i^n.\alpha \neq \texttt{accepted}$ 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 $\pi_i^n.\alpha \neq \texttt{accepted}$ 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 $sync \leftarrow \texttt{false}$
if $sync = \texttt{false}$ then return M
return $\perp$

Figure 5.3 – The Encrypt and Decrypt oracles in the 3-ACCE security experiment. StAE is the stateful authenticated encryption scheme used to establish the secure tunnel. The counters u and v are initialised to 0, and $sync$ to `true` at the beginning of every session. In case π_i^n does not have a partner when answering a Decrypt query, then $sync = \texttt{false}$.

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. The instance's state is set to $\pi_i^n.\alpha = \texttt{running}$.
- $\mathsf{Send}(\pi_i^n, M)$: 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 $\pi_i^n.\alpha = \texttt{accepted}$. (Send queries in an accepting state are handled by the 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 $\pi_i^n.\alpha \neq \texttt{accepted}$, 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 $\pi_i^n.\alpha \neq \texttt{accepted}$, then π_i^n returns $\perp$). Figure 5.3 depicts this oracle.

5.3.3 Security Definitions

Partnership. In order to define the partnership between two instances, we use the notion of matching conversations (see Definition 2.1). 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 (based on the left-or-right indistinguishability variant, see Chapter 2, Section 2.2.5). 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 participates 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. [BBF+17], 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 the EA security experiment, the adversary is successful if, when it terminates, there exists an instance that *maliciously accepts* according to the following definition.

Definition 5.5 (EA)**.** *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*

- $\pi_i^n.\alpha = \texttt{accepted}$ *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 $\pi_k^\ell \in P_k.\mathsf{Instances} \mid \pi_k^\ell.\mathsf{km} = \pi_i^n.\mathsf{ck}$.

NS adversary – 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)
- $\pi_j^u.\alpha = \texttt{accepted}$ *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)
- $\pi_j^v.\alpha = \texttt{accepted}$ *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 $\pi_i^n \mid (\pi_i^n.\mathsf{parent} \in \mathcal{E} \wedge \pi_i^n.\mathsf{pid} = P_k \wedge \pi_i^n.\mathsf{ck} = \pi_j^v.\mathsf{km})$.

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

- $\pi_k^\ell.\alpha = \texttt{accepted}$ *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:

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

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 it guesses b. That is, it 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 5.6 (CS)**.** *An adversary* $\mathcal{A}$ *breaks the channel security if it terminates the channel security game with a tuple* (π_i^n, b) *such that*

- $\pi_i^n.\alpha = \texttt{accepted}$
- *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

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

Definition 5.7 (3-ACCE-security)**.** *A 3-party protocol* Π *is 3-ACCE-secure if* Π *satisfies correctness, and for all probabilistic polynomial time adversaries* $\mathcal{A}$*,* $\mathsf{adv}_{\Pi}^{\text{ent-auth}}(\mathcal{A})$ *and* $\mathsf{adv}_{\Pi}^{\text{chan-sec}}(\mathcal{A})$ *are a negligible function of the security parameter.*

5.3.4 Comparison with Existing Models

The 3-ACCE security notion we propose takes inspiration from that of Bhargavan et al. [BBF+17], 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 ACCE model introduced by Jager et al. [JKSS11] to prove the security of TLS 1.2 [DR08] in DHE mode, and used by Kohlar et al. to prove the security of TLS 1.2 in RSA and DH modes [KSS13]. Our model follows the same execution environment and adversarial model, and reuse the corresponding notation to deal with the entity authentication and the channel security properties. Yet we relax the stateful length-hiding AEAD (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 the 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.[1] 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 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.[2] 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.

[1] This property is called *accountability* in [BBF+17].

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

The ACCE-AP model of Bhargavan et al. [BBD+18] 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 [BBD+18] observe that their model is complex and achieves limited record-layer guarantees in multi-middlebox setting.

5.4 Security Proofs in the 3-ACCE Model

5.4.1 3-ACCE Security for a Generic 3-party Protocol

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 5.3.2 and 5.3.3.

5.4.1.1 Description of the Generic 3-party Protocol

Figure 5.4 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 5.5.

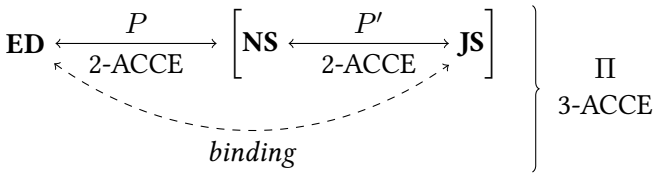


Figure 5.4 – 3-ACCE protocol Π

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 [ET05], SRP [Wu00], and SIGMA-R [Kra03] can be instances of P. As we will see in Section 5.4.2, LoRaWAN 1.1 is such another instance.

5.4.1.2 Main Theorem and Sketch of Proof

Based on the security of P and P', we show that Π is 3-ACCE-secure according to Definition 5.7.

Theorem 5.1. *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, and for any probabilistic polynomial time*

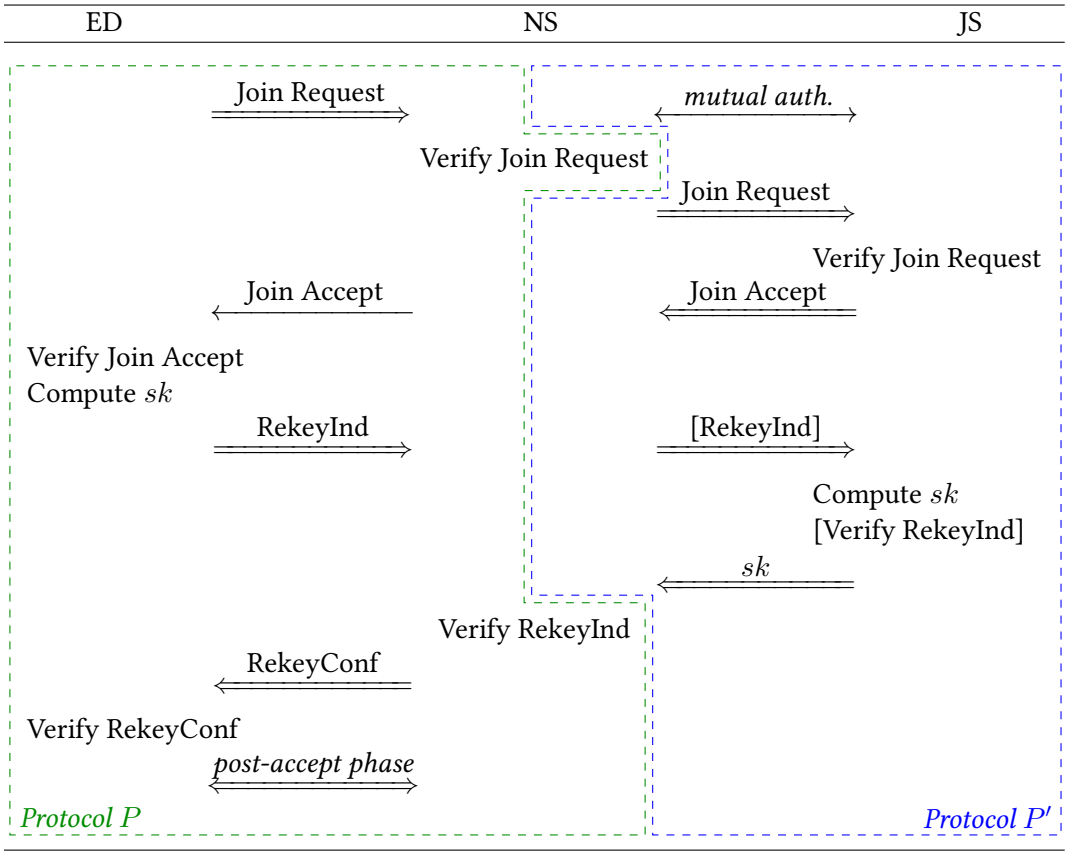


Figure 5.5 – Correct execution of protocol Π, made of P (left) and P' (right) components. Double line arrows indicate the use of the secure channel keys.

adversary $\mathcal{A}$ in the 3-ACCE security experiment against Π

$$\begin{aligned}
\mathsf{adv}^{\text{ent-auth}}_{\Pi}(\mathcal{A}) \;\leq\; & n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(2\mathsf{adv}^{\text{chan-sec}}_{P}(\mathcal{B}_0) + 3\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) + 2p_{jr} + 2p_{ja}\right. \\
& \left. + \mathsf{adv}^{\text{ent-auth}}_{P',\text{client}}(\mathcal{B}_1) + \mathsf{adv}^{\text{ent-auth}}_{P',\text{server}}(\mathcal{B}_1)\right) \\
& + n_{\mathsf{E}} \left(n_{\mathsf{J}} \cdot \mathsf{adv}^{\text{ent-auth}}_{P,\text{client}}(\mathcal{B}_0) + n_{\mathsf{N}} \cdot \mathsf{adv}^{\text{ent-auth}}_{P,\text{server}}(\mathcal{B}_0)\right) \\
& + n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(3\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\text{ent-auth}}_{P',\text{client}}(\mathcal{B}_1) + \mathsf{adv}^{\text{ent-auth}}_{P',\text{server}}(\mathcal{B}_1)\right) \\
\mathsf{adv}^{\text{chan-sec}}_{\Pi}(\mathcal{A}) \;\leq\; & n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\mathsf{adv}^{\text{chan-sec}}_{P}(\mathcal{B}_0) + 3\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1)\right) + \mathsf{adv}^{\text{ent-auth}}_{\Pi}(\mathcal{A})
\end{aligned}$$

where n_{E}, n_{N}, and n_{J} are respectively the number of ED, NS, and JS parties, $\mathcal{B}_0$ is an adversary against the 2-ACCE-security of P, and $\mathcal{B}_1$ is an adversary against the 2-ACCE-security of P'.

We give here a sketch of proof for Theorem 5.1, and provide a full proof in Section 5.4.3.1.

Sketch of proof. 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 ED accept without NS or JS being involved. Hence the adversary can first try to impersonate NS to 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 $\mathsf{adv}^{\text{ent-auth}}_{P,\text{client}}(\mathcal{B}_0)$. The adversary can also try to bypass

the intermediate NS in order to get from 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 JS, that is to break the EA-security of P' (corresponding to an advantage $\mathsf{adv}^{\text{ent-auth}}_{P',\text{server}}(\mathcal{B}_1)$). 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 NS different than the one computed by ED. This implies either forging such a message, or getting the keys used to compute it and transmitted by JS to NS. We reduce both possibilities to the channel security with respect to P on the one hand ($\mathsf{adv}^{\text{chan-sec}}_{P}(\mathcal{B}_0)$), and to the channel security with respect to P' on the other hand ($\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1)$).

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 JS upon reception of the Join Request message computed by 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

$$\begin{aligned}\mathsf{adv}^{\text{ent-auth}}_{\Pi,\mathsf{E}}(\mathcal{A}) \quad \leq \quad & n_{\mathsf{E}} \cdot n_{\mathsf{J}_\mathsf{E}} \left(n_{\mathsf{N}} \cdot \left(\mathsf{adv}^{\text{ent-auth}}_{P',\text{server}}(\mathcal{B}_1) + \mathsf{adv}^{\text{chan-sec}}_{P}(\mathcal{B}_0) + \mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) \right) \right. \\ & \left. + \mathsf{adv}^{\text{ent-auth}}_{P,\text{client}}(\mathcal{B}_0) \right)\end{aligned}$$

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 ED to 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 $\mathsf{adv}^{\text{ent-auth}}_{P,\text{server}}(\mathcal{B}_0)$. Then the adversary can try to impersonate a legitimate JS to NS, in order to preclude the involvement of JS in the protocol run. This corresponds to an advantage $\mathsf{adv}^{\text{ent-auth}}_{P',\text{client}}(\mathcal{B}_1)$.

The only cryptographic operation that NS does in order to accept is verifying the RekeyInd message it gets from ED with the keys provided by JS. Therefore the adversary is successful if, on the one hand, it provides some keys sk to NS (through the P' secure channel), and, on the other hand, it 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.[3] This is possible if the adversary forges a valid P' application message carrying the keys sk it has chosen. This can be reduced to the channel security with respect to P', which corresponds to the advantage $\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1)$.

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).[4] 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 JS to NS through a secure channel with respect to P'. We reduce either possibility respectively to the channel security with respect to P ($\mathsf{adv}^{\text{chan-sec}}_{P}(\mathcal{B}_0)$), and to the channel security with respect to P' ($\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1)$). 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,

[3] The adversary sends first a fake Join Request message to NS (random MAC tag, correct counter cnt_E). Then the adversary sends a random Join Accept to NS followed by session keys sk of its choice, and a RekeyInd message computed under sk (as depicted in Section 5.3.1 by Figure 5.2).

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

the advantage of an NS adversary in winning through condition (a) is bounded by

$$p_a \leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(\mathsf{adv}^{\text{ent-auth}}_{P,\text{server}}(\mathcal{B}_0) + n_{\mathsf{J_E}} \left(\mathsf{adv}^{\text{ent-auth}}_{P',\text{client}}(\mathcal{B}_1) + 2\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\text{chan-sec}}_{P}(\mathcal{B}_0) \right) \right).$$

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 $\mathsf{adv}^{\text{ent-auth}}_{P',\text{client}}(\mathcal{B}_1)$. Then the adversary can proceed as under condition (a). That is providing to NS some keys sk of its 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 $\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1)$. 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.[5] We can reduce the latter to the channel security of P' ($\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1)$).

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 NS and 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 5.4.2.1).

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

$$p_b \leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\mathsf{adv}^{\text{ent-auth}}_{P',\text{client}}(\mathcal{B}_1) + 2\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) + n_{\mathsf{E_J}} \left(p_{jr} + p_{ja} \right) \right).$$

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

$$\begin{aligned} \mathsf{adv}^{\text{ent-auth}}_{\Pi,\mathsf{N}}(\mathcal{A}) &\leq p_a + p_b \\ &\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J_E}} \cdot \left(\mathsf{adv}^{\text{chan-sec}}_{P}(\mathcal{B}_0) + 2\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\text{ent-auth}}_{P',\text{client}}(\mathcal{B}_1) \right) \right. \\ &\qquad \left. + \mathsf{adv}^{\text{ent-auth}}_{P,\text{server}}(\mathcal{B}_0) \right) \\ &\quad + n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\mathsf{adv}^{\text{ent-auth}}_{P',\text{client}}(\mathcal{B}_1) + 2\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) + n_{\mathsf{E_J}} \left(p_{jr} + p_{ja} \right) \right) \end{aligned}$$

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

JS adversary. In this setting, the adversary can first try to impersonate NS to JS, which corresponds to an advantage $\mathsf{adv}^{\text{ent-auth}}_{P',\text{server}}(\mathcal{B}_1)$. Then, in order to have that NS and 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' ($\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1)$). 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 ED and 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

$$\mathsf{adv}^{\text{ent-auth}}_{\Pi,\mathsf{J}}(\mathcal{A}) \leq n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(\mathsf{adv}^{\text{ent-auth}}_{P',\text{server}}(\mathcal{B}_1) + \mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) + n_{\mathsf{E_J}} \left(p_{jr} + p_{ja} \right) \right).$$

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.

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

This leads to an advantage equal to $\mathsf{adv}_{\Pi}^{\text{ent-auth}}(\mathcal{A})$. This leaves two possibilities in order for the adversary to be successful: either it targets directly the secure channel between ED and NS, or it targets the secure channel between NS and JS. We can reduce the latter possibility to the CS-security of P' corresponding to an advantage $\mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)$. Regarding the former possibility, the adversary can first try to get the P session keys (sk) sent by JS to NS (which we reduce to the channel security with respect to P' leading to an advantage $\mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)$). Then the adversary can try to break the channel security with respect to P ($\mathsf{adv}_{P}^{\text{chan-sec}}(\mathcal{B}_0)$). We have also to take into account that the session keys sk are sent by JS to 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 $\mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)$).
Accounting that the reduction must guess the identity of the parties involved, the advantage of the adversary is bounded by

$$\mathsf{adv}_{\Pi}^{\text{chan-sec}}(\mathcal{A}) \leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\mathsf{adv}_{P}^{\text{chan-sec}}(\mathcal{B}_0) + 3\mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)\right) + \mathsf{adv}_{\Pi}^{\text{ent-auth}}(\mathcal{A}).$$

□

5.4.2 3-ACCE Security with LoRaWAN 1.1

In this section, we use the generic result of Section 5.4.1, and apply it to LoRaWAN 1.1. 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 5.1, (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 Chapter 3, Section 3.5, 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. Another reason to opt for such a deployment is that LoRaWAN does not provide end-to-end data integrity between ED and AS which leads to trivial attacks (see Section 3.6.5). Thus, AS is in fact merely a functionality handled by NS, and the latter is given the four session keys K_a^e, K_c^e, $K_c^{i_1}$, $K_c^{i_2}$ (see Section 3.5.3). Hence, we instantiate LoRaWAN accordingly: our protocol is made of three active entities (ED, NS, JS) which the different cryptographic operations are attributed to. This makes the attack scenarios presented in Sections 3.6.6 and 3.6.7 ineffective.

5.4.2.1 2-party Protocol P in LoRaWAN 1.1 is 2-ACCE Secure

In this section, we use the ACCE security model described in Chapter 2 (we rename 2-ACCE this model in order to distinguish from the 3-ACCE model). 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 $P_k \in \mathcal{J}$. The master key mk is defined as $mk = (MK_1, MK_2)$ where MK_1 and MK_2 are respectively ED's communication master key, and application master key (see Section 3.5.2).

Let $P_{LoRaWAN}$ correspond to the messages exchanged, and the operations done between a client (ED) and a server (NS-JS) based on LoRaWAN 1.1. We claim with the following theorem that the protocol $P_{LoRaWAN}$ is a secure 2-ACCE protocol. 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 in LoRaWAN 1.1.

Theorem 5.2. *Under the assumption that* $\mathsf{StAE}_{\mathsf{client}}$ *and* $\mathsf{StAE}_{\mathsf{server}}$ *are sAE-secure,* $P_{LoRaWAN}$ *is a secure 2-ACCE protocol, and for any probabilistic polynomial time adversary* $\mathcal{A}$ *in the 2-ACCE security experiment against* $P_{LoRaWAN}$

$$\begin{aligned}
\mathsf{adv}_P^{\mathsf{ent\text{-}auth}}(\mathcal{A}) &\leq q\Big[(n_{\mathsf{C}}+n_{\mathsf{S}})\Big(\mathsf{adv}_{\mathsf{MAC}}^{\mathsf{prf}}(\mathcal{B}_0)+2\mathsf{adv}_{\mathsf{AES}}^{\mathsf{prf}}(\mathcal{B}_1)\Big)+n_{\mathsf{S}}\cdot\mathsf{adv}_{\mathsf{StAE}_{\mathsf{client}}}^{\mathsf{sae}}(\mathcal{B}_3)\\
&\quad+n_{\mathsf{C}}\left(\mathsf{adv}_{\mathsf{AES}}^{\mathsf{prp}}(\mathcal{B}_2)+\mathsf{adv}_{\mathsf{StAE}_{\mathsf{server}}}^{\mathsf{sae}}(\mathcal{B}_3)\right)+2^{-\mu}\left(n_{\mathsf{C}}\cdot(1-2^{-\beta})+n_{\mathsf{S}}\right)\Big]\\
\mathsf{adv}_P^{\mathsf{chan\text{-}sec}}(\mathcal{A}) &\leq q^2\cdot n_{\mathsf{C}}\cdot n_{\mathsf{S}}\Big(\mathsf{adv}_{\mathsf{StAE}_{\mathsf{client}}}^{\mathsf{sae}}(\mathcal{B}_3)+\mathsf{adv}_{\mathsf{StAE}_{\mathsf{server}}}^{\mathsf{sae}}(\mathcal{B}_3)+2\mathsf{adv}_{\mathsf{AES}}^{\mathsf{prf}}(\mathcal{B}_1)\Big)\\
&\quad+\mathsf{adv}_P^{\mathsf{ent\text{-}auth}}(\mathcal{A})
\end{aligned}$$

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,* β *is the bit length of the counter* cnt_J*, and* $\mathcal{B}_0$ *is an adversary against the PRF-security of* MAC*,* $\mathcal{B}_1$ *an adversary against the PRF-security of* AES*,* $\mathcal{B}_2$ *an adversary against the PRP-security of* AES*, and* $\mathcal{B}_3$ *an adversary against the sAE-security of* StAE*.*

We give here a sketch of proof for Theorem 5.2, and provide a full proof in Section 5.4.3.3.

Sketch of proof. We consider the 2-ACCE security model of Jager et al. [JKSS11] described in Chapter 2, Section 2.3.1, 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 [BWJM97]).

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 $\mathsf{adv}_{\mathsf{AES}}^{\mathsf{prf}}(\mathcal{B}_1)$, $\mathsf{adv}_{\mathsf{MAC}}^{\mathsf{prf}}(\mathcal{B}_0)$, and $\mathsf{adv}_{\mathsf{AES}}^{\mathsf{prp}}(\mathcal{B}_2)$. 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 $\Pr[\text{forgery Join Accept}] \leq p_{ja} = \mathsf{adv}_{\mathsf{AES}}^{\mathsf{prf}}(\mathcal{B}_1)+\mathsf{adv}_{\mathsf{MAC}}^{\mathsf{prf}}(\mathcal{B}_0)+\mathsf{adv}_{\mathsf{AES}}^{\mathsf{prp}}(\mathcal{B}_2)+2^{-\mu}(1-2^{-\beta})$. 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 $\mathsf{adv}_{\mathsf{AES}}^{\mathsf{prf}}(\mathcal{B}_1)+\mathsf{adv}_{\mathsf{StAE}_{\mathsf{server}}}^{\mathsf{sae}}(\mathcal{B}_3)$). 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

$$\begin{aligned}
\mathsf{adv}_{P,\mathsf{client}}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) &\leq q\cdot n_{\mathsf{C}}\Big(\mathsf{adv}_{\mathsf{StAE}_{\mathsf{server}}}^{\mathsf{sae}}(\mathcal{B}_3)+2^{-(\mu+\beta)}(2^\beta-1)+\mathsf{adv}_{\mathsf{AES}}^{\mathsf{prp}}(\mathcal{B}_2)+\mathsf{adv}_{\mathsf{MAC}}^{\mathsf{prf}}(\mathcal{B}_0)\\
&\quad+2\mathsf{adv}_{\mathsf{AES}}^{\mathsf{prf}}(\mathcal{B}_1)\Big)
\end{aligned}$$

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 $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0)$, and $2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1)$. 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 $\Pr[\text{forgery Join Request}] \leq p_{jr} = \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0)+2^{-\mu}$. 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 $\mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{client}}}(\mathcal{B}_3)$). 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

$$\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P,\mathsf{server}}(\mathcal{A}) \leq q \cdot n_{\mathsf{S}} \left(\mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{client}}}(\mathcal{B}_3) + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) + 2^{-\mu} + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) \right)$$

In addition, we have also

$$\begin{array}{rcccl} \Pr[\text{forgery Join Request}] & \leq & p_{jr} & = & \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) + 2^{-\mu} \\ \Pr[\text{forgery Join Accept}] & \leq & p_{ja} & = & \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) + \mathsf{adv}^{\mathsf{prp}}_{\mathsf{AES}}(\mathcal{B}_2) \\ & & & & +2^{-(\mu+\beta)} \cdot (2^{\beta} - 1) \end{array}$$

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 $\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P}(\mathcal{A})$. 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 $2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1)$. 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 $\mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{client}}}(\mathcal{B}_3) + \mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{server}}}(\mathcal{B}_3)$. 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

$$\mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P}(\mathcal{A}) \leq q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}} \left(\mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{client}}}(\mathcal{B}_3) + \mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{server}}}(\mathcal{B}_3) + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) \right) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P}(\mathcal{A})$$

□

5.4.2.2 Meeting 3-ACCE Security

As seen in Chapter 3, Section 3.6, and also exhibited by Theorem 5.2, 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 5.2. Therefore, applying the recommendations presented in Section 3.7 in order to mitigate these vulnerabilities, we modify LoRaWAN 1.1 in the following way in order to yield $P_{LoRaWAN}$.

- We demand that the size μ of the MAC output be high enough so that the security bounds $\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P}$ and $\mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P}$ be tight (recommendation R2).
- We slightly change the behaviour of JS as follows: JS verifies entirely the Join Request message (including ED's counter cnt_E), and the RekeyInd message. It sends the session keys sk to 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, JS has no guarantee that ED successfully completes the protocol (it is enough for the adversary to drop or alter the Join Accept message in order for ED to not accept).
The components surrounded with brackets in Figure 5.5 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 (recommendation R3). 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.
- We require that all entities implement version 1.1 (including NS) so that no fallback to LoRaWAN 1.0 be possible, and the vulnerabilities of that version be avoided (recommendation R4).
- In addition, we demand that ED apply recommendation R1 which aims at precluding a counter exhaustion (see Section 3.6.1).

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

Now we define the companion security protocol $P'_{LoRaWAN}$ that is used between NS and JS. As explained in Section 5.3.1, the careful choice of this protocol is crucial to the overall security of a LoRaWAN network. Indeed, the attack scenario described in Section 5.3.1 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 [DR08] in DHE, or RSA mode, with mutual authentication, and instantiated with AEAD encryption schemes such as AES-GCM, AES-CCM [McG08], or ChaCha20-Poly1305 [NL15]. TLS 1.2 is known to be 2-ACCE-secure [JKSS11; KSS13]. Alternatively, $P'_{LoRaWAN}$ can be defined as TLS 1.3 [Res18] 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 [DFGS15]. 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 5.1, we obtain the 3-ACCE-security of our adapted version of LoRaWAN 1.1.

5.4.3 Extended Security Proofs

5.4.3.1 Extended Security Proof for Π

In this section, we give the full proof of Theorem 5.1. We proceed through a sequence of games [Sho04; BR04] between a challenger and an adversary $\mathcal{A}$.

Entity authentication. First we consider the entity authentication experiment described in Section 5.3.3.

Proof. Let $\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{\Pi,\mathsf{X}}(\mathcal{A})$ 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 $\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{\Pi}(\mathcal{A}) \leq$

$\mathsf{adv}^{\text{ent-auth}}_{\Pi,\mathsf{E}}(\mathcal{A}) + \mathsf{adv}^{\text{ent-auth}}_{\Pi,\mathsf{N}}(\mathcal{A}) + \mathsf{adv}^{\text{ent-auth}}_{\Pi,\mathsf{J}}(\mathcal{A})$.

ED adversary. Let E_i be the event that the adversary succeeds in making an instance maliciously accept during Game i, where the instance belongs to 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 5.3.3 when the adversary targets ED. Therefore we have that

$$\Pr[E_0] = \mathsf{adv}^{\text{ent-auth}}_{\Pi,\mathsf{E}}(\mathcal{A}).$$

Game 1. In this game, the challenger aborts the experiment if it 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 in $\mathcal{N}$ 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 $\mathsf{adv}^{\text{ent-auth}}_{P,\mathsf{client}}(\mathcal{B}_0)$ for a client adversary $\mathcal{B}_0$ 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

$$\Pr[E_1] \leq \Pr[E_2] + \mathsf{adv}^{\text{ent-auth}}_{P,\mathsf{client}}(\mathcal{B}_0).$$

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 it 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 $\mathcal{B}_1$ able to break the EA-security of P' when targeting the server side. Therefore

$$\Pr[E_3] \leq \Pr[E_4] + \mathsf{adv}^{\text{ent-auth}}_{P',\mathsf{server}}(\mathcal{B}_1).$$

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). Now 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

$$\Pr[E_4] \leq \Pr[E_5] + \mathsf{adv}_P^{\mathsf{chan\text{-}sec}}(\mathcal{B}_0) + \mathsf{adv}_{P'}^{\mathsf{chan\text{-}sec}}(\mathcal{B}_1).$$

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. Hence

$$\Pr[E_5] = 0.$$

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

$$\begin{aligned}
\mathsf{adv}_{\Pi,\mathsf{E}}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) &= \Pr[E_0] \\
&= n_{\mathsf{E}} \times n_{\mathsf{J_E}} \times \Pr[E_1] \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{J_E}} \left(\Pr[E_2] + \mathsf{adv}_{P,\mathsf{client}}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_0)\right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{J_E}} \left(n_{\mathsf{N}} \cdot \Pr[E_3] + \mathsf{adv}_{P,\mathsf{client}}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_0)\right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{J_E}} \left(n_{\mathsf{N}} \cdot \left(Pr[E_4] + \mathsf{adv}_{P',\mathsf{server}}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_1)\right) + \mathsf{adv}_{P,\mathsf{client}}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_0)\right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{J_E}} \left(n_{\mathsf{N}} \cdot \left(Pr[E_5] + \mathsf{adv}_P^{\mathsf{chan\text{-}sec}}(\mathcal{B}_0) + \mathsf{adv}_{P'}^{\mathsf{chan\text{-}sec}}(\mathcal{B}_1)\right.\right. \\
&\qquad \left.\left. + \mathsf{adv}_{P',\mathsf{server}}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_1)\right) \right. \\
&\qquad \left. + \mathsf{adv}_{P,\mathsf{client}}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_0)\right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{J_E}} \left(n_{\mathsf{N}} \cdot \left(\mathsf{adv}_P^{\mathsf{chan\text{-}sec}}(\mathcal{B}_0) + \mathsf{adv}_{P'}^{\mathsf{chan\text{-}sec}}(\mathcal{B}_1) + \mathsf{adv}_{P',\mathsf{server}}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_1)\right)\right. \\
&\qquad \left. + \mathsf{adv}_{P,\mathsf{client}}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_0)\right)
\end{aligned}$$

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 $\mathsf{adv}_{\Pi,\mathsf{N}}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) \leq p_a + p_b$. 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 5.3.3 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 it 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 $\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P,\mathsf{server}}(\mathcal{B}_0)$. Therefore

$$\Pr[E_1^a] \leq \Pr[E_2^a] + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P,\mathsf{server}}(\mathcal{B}_0).$$

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 $\pi_j^u.\alpha = \texttt{accepted}$ 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 $\mathcal{J}$ which 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 its choice to P_j, it 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 the adversary). We preclude such events in the subsequent sequence of games.

But, before considering this case, the challenger aborts the experiment if it does not guess which party $P_k \in \mathcal{J}$ is the intended partner of P_i (during the execution of protocol P). There are 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

$$\Pr[E_3^a] \leq \Pr[E_4^a] + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P',\mathsf{client}}(\mathcal{B}_1).$$

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

$$\Pr[E_4^a] \leq \Pr[E_5^a] + \mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P'}(\mathcal{B}_1).$$

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 necessarily receives 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

$$\Pr[E_5^a] \leq \Pr[E_6^a] + \mathsf{adv}_P^{\text{chan-sec}}(\mathcal{B}_0) + \mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1).$$

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

$$\begin{aligned}
p_a &= \Pr[E_0^a] \\
&= n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot \Pr[E_1^a] \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(\Pr[E_2^a] + \mathsf{adv}_{P,\text{server}}^{\text{ent-auth}}(\mathcal{B}_0)\right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J_E}} \cdot \Pr[E_3^a] + \mathsf{adv}_{P,\text{server}}^{\text{ent-auth}}(\mathcal{B}_0)\right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J_E}} \cdot \left(\Pr[E_4^a] + \mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)\right) + \mathsf{adv}_{P,\text{server}}^{\text{ent-auth}}(\mathcal{B}_0)\right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J_E}} \cdot \left(\Pr[E_5^a] + \mathsf{adv}_{P',\text{client}}^{\text{ent-auth}}(\mathcal{B}_1) + \mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)\right) + \mathsf{adv}_{P,\text{server}}^{\text{ent-auth}}(\mathcal{B}_0)\right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \Big(n_{\mathsf{J_E}} \cdot \Big(\Pr[E_6^a] + \mathsf{adv}_P^{\text{chan-sec}}(\mathcal{B}_0) + \mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1) + \mathsf{adv}_{P',\text{client}}^{\text{ent-auth}}(\mathcal{B}_1) \\
&\qquad\qquad + \mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)\Big) \\
&\qquad + \mathsf{adv}_{P,\text{server}}^{\text{ent-auth}}(\mathcal{B}_0)\Big) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J_E}} \cdot \left(\mathsf{adv}_P^{\text{chan-sec}}(\mathcal{B}_0) + \mathsf{adv}_{P',\text{client}}^{\text{ent-auth}}(\mathcal{B}_1) + 2\mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)\right) + \mathsf{adv}_{P,\text{server}}^{\text{ent-auth}}(\mathcal{B}_0)\right)
\end{aligned}$$

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 5.3.3 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 it 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 its choice, it 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

$$\Pr[E_1^b] \leq \Pr[E_2^b] + \mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P'}(\mathcal{B}_1).$$

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

$$\Pr[E_2^b] \leq \Pr[E_3^b] + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P',\mathsf{client}}(\mathcal{B}_1).$$

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 $\pi_j^v.\alpha = \texttt{accepted}$ 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

$$\Pr[E_3^b] \leq \Pr[E_4^b] + \mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P'}(\mathcal{B}_1).$$

So far, we have shown that for any instance $\pi_j^v \in P_j.\mathsf{Instances}$ such that $\pi_j^v.\alpha = \texttt{accepted}$ 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 it 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.[6] 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 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

$$\begin{aligned}
p_b &= \Pr[E_0^b] \\
&= n_{\mathsf{N}} \cdot n_{\mathsf{J}} \cdot \Pr[E_1^b] \\
&\leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\Pr[E_2^b] + \mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)\right) \\
&\leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\Pr[E_3^b] + \mathsf{adv}_{P',\mathsf{client}}^{\text{ent-auth}}(\mathcal{B}_1) + \mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)\right) \\
&\leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\Pr[E_4^b] + \mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1) + \mathsf{adv}_{P',\mathsf{client}}^{\text{ent-auth}}(\mathcal{B}_1) + \mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)\right) \\
&\leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(n_{\mathsf{E_J}} \cdot \Pr[E_5^b] + \mathsf{adv}_{P',\mathsf{client}}^{\text{ent-auth}}(\mathcal{B}_1) + 2\mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)\right) \\
&\leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(n_{\mathsf{E_J}} \left(\Pr[E_6^b] + p_{jr}\right) + \mathsf{adv}_{P',\mathsf{client}}^{\text{ent-auth}}(\mathcal{B}_1) + 2\mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)\right) \\
&\leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(n_{\mathsf{E_J}} \left(\Pr[E_7^b] + p_{jr} + p_{ja}\right) + \mathsf{adv}_{P',\mathsf{client}}^{\text{ent-auth}}(\mathcal{B}_1) + 2\mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)\right) \\
&\leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(n_{\mathsf{E_J}} \left(p_{jr} + p_{ja}\right) + \mathsf{adv}_{P',\mathsf{client}}^{\text{ent-auth}}(\mathcal{B}_1) + 2\mathsf{adv}_{P'}^{\text{chan-sec}}(\mathcal{B}_1)\right)
\end{aligned}$$

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

Therefore we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{\Pi,\mathsf{N}}(\mathcal{A}) &\leq p_a + p_b \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J}_{\mathsf{E}}} \cdot \left(\mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P}(\mathcal{B}_0) + 2\mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P',\mathsf{client}}(\mathcal{B}_1) \right) \right. \\
&\qquad \left. + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P,\mathsf{server}}(\mathcal{B}_0) \right) \\
&\quad + n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(n_{\mathsf{E}_{\mathsf{J}}} \left(p_{jr} + p_{ja} \right) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P',\mathsf{client}}(\mathcal{B}_1) + 2\mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P'}(\mathcal{B}_1) \right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P,\mathsf{server}}(\mathcal{B}_0) \\
&\quad + (n_{\mathsf{E}} + 1) \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \cdot \left(\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P',\mathsf{client}}(\mathcal{B}_1) + 2\mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P'}(\mathcal{B}_1) \right) \\
&\quad + n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(p_{jr} + p_{ja} + \mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P}(\mathcal{B}_0) \right)
\end{aligned}$$

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

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 5.3.3 when the adversary targets JS. Therefore we have that

$$\Pr[E_0] = \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{\Pi,\mathsf{J}}(\mathcal{A}).$$

Game 1. In this game, the challenger aborts the experiment if it 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

$$\Pr[E_1] \leq \Pr[E_2] + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P',\mathsf{server}}(\mathcal{B}_1).$$

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 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

$$\Pr[E_2] \leq \Pr[E_3] + \mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P'}(\mathcal{B}_1).$$

Game 4. Now, for each instance $\pi_k^\ell \in P_k.\mathsf{Instances}$ such that $\pi_k^\ell.\alpha = \texttt{accepted}$, there is a unique instance $\pi_j^v \in P_j.\mathsf{Instances}$ such that $\pi_k^\ell.\mathsf{sid} = \pi_j^v.\mathsf{sid}$.

In this game, the challenger aborts the experiment if it 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

$$\begin{aligned}
\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{\Pi,\mathsf{J}}(\mathcal{A}) &= \Pr[E_0] \\
&= n_{\mathsf{J}} \cdot n_{\mathsf{N}} \cdot \Pr[E_1] \\
&\leq n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(\Pr[E_2] + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P',\mathsf{server}}(\mathcal{B}_1)\right) \\
&\leq n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(\Pr[E_3] + \mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P',\mathsf{server}}(\mathcal{B}_1)\right) \\
&\leq n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{E_J}} \cdot \Pr[E_4] + \mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P',\mathsf{server}}(\mathcal{B}_1)\right) \\
&\leq n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{E_J}} \left(\Pr[E_5] + p_{jr}\right) + \mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P',\mathsf{server}}(\mathcal{B}_1)\right) \\
&\leq n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{E_J}} \left(\Pr[E_6] + p_{jr} + p_{ja}\right) + \mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P',\mathsf{server}}(\mathcal{B}_1)\right) \\
&\leq n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{E_J}} \left(p_{jr} + p_{ja}\right) + \mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P',\mathsf{server}}(\mathcal{B}_1)\right)
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
\mathsf{adv}^{\text{ent-auth}}_{\Pi}(\mathcal{A}) &\leq \mathsf{adv}^{\text{ent-auth}}_{\Pi,\mathsf{E}}(\mathcal{A}) + \mathsf{adv}^{\text{ent-auth}}_{\Pi,\mathsf{N}}(\mathcal{A}) + \mathsf{adv}^{\text{ent-auth}}_{\Pi,\mathsf{J}}(\mathcal{A}) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{J}_{\mathsf{E}}} \left(n_{\mathsf{N}} \cdot \left(\mathsf{adv}^{\text{chan-sec}}_{P}(\mathcal{B}_0) + \mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\text{ent-auth}}_{P',\text{server}}(\mathcal{B}_1)\right)\right. \\
&\qquad \left. + \mathsf{adv}^{\text{ent-auth}}_{P,\text{client}}(\mathcal{B}_0)\right) \\
&\quad + n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J}_{\mathsf{E}}} \cdot \left(\mathsf{adv}^{\text{chan-sec}}_{P}(\mathcal{B}_0) + 2\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\text{ent-auth}}_{P',\text{client}}(\mathcal{B}_1)\right)\right. \\
&\qquad \left. + \mathsf{adv}^{\text{ent-auth}}_{P,\text{server}}(\mathcal{B}_0)\right) \\
&\quad + n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(n_{\mathsf{E}_{\mathsf{J}}} \left(p_{jr} + p_{ja}\right) + \mathsf{adv}^{\text{ent-auth}}_{P',\text{client}}(\mathcal{B}_1) + 2\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1)\right) \\
&\quad + n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{E}_{\mathsf{J}}} \left(p_{jr} + p_{ja}\right) + \mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\text{ent-auth}}_{P',\text{server}}(\mathcal{B}_1)\right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(2\mathsf{adv}^{\text{chan-sec}}_{P}(\mathcal{B}_0) + 3\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) + 2p_{jr} + 2p_{ja}\right. \\
&\qquad \left. + \mathsf{adv}^{\text{ent-auth}}_{P',\text{client}}(\mathcal{B}_1) + \mathsf{adv}^{\text{ent-auth}}_{P',\text{server}}(\mathcal{B}_1)\right) \\
&\quad + n_{\mathsf{E}} \left(n_{\mathsf{J}} \cdot \mathsf{adv}^{\text{ent-auth}}_{P,\text{client}}(\mathcal{B}_0) + n_{\mathsf{N}} \cdot \mathsf{adv}^{\text{ent-auth}}_{P,\text{server}}(\mathcal{B}_0)\right) \\
&\quad + n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(3\mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1) + \mathsf{adv}^{\text{ent-auth}}_{P',\text{client}}(\mathcal{B}_1) + \mathsf{adv}^{\text{ent-auth}}_{P',\text{server}}(\mathcal{B}_1)\right)
\end{aligned}$$

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

Channel security. Now we prove the channel security property.

Proof. Let $\mathsf{adv}^{\text{chan-sec}}_{\Pi}(\mathcal{A})$ 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 5.3.3. Therefore

$$\Pr[E_0] = \frac{1}{2} + \mathsf{adv}_0 = \frac{1}{2} + \mathsf{adv}^{\text{chan-sec}}_{\Pi}(\mathcal{A}).$$

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

$$\mathsf{adv}_0 \leq \mathsf{adv}_1 + \mathsf{adv}^{\text{ent-auth}}_{\Pi}(\mathcal{A}).$$

Game 2. In this game, the challenger aborts the experiment if it 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 $\pi^v_j.\mathsf{b}$ or $\pi^\ell_k.\mathsf{b}$ (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

$$\mathsf{adv}_2 \leq \mathsf{adv}_3 + \mathsf{adv}^{\text{chan-sec}}_{P'}(\mathcal{B}_1).$$

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

$$\mathsf{adv}_3 \leq \mathsf{adv}_4 + \mathsf{adv}_{P'}^{\mathsf{chan\text{-}sec}}(\mathcal{B}_1).$$

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 [BDJR97]. 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

$$\mathsf{adv}_4 \leq \mathsf{adv}_5 + \mathsf{adv}_{P'}^{\mathsf{chan\text{-}sec}}(\mathcal{B}_1).$$

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

$$\mathsf{adv}_5 \leq \mathsf{adv}_P^{\mathsf{chan\text{-}sec}}(\mathcal{B}_0).$$

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

$$\begin{aligned}
\mathsf{adv}_\Pi^{\mathsf{chan\text{-}sec}}(\mathcal{A}) &= \mathsf{adv}_0 \\
&\leq \mathsf{adv}_1 + \mathsf{adv}_\Pi^{\mathsf{ent\text{-}auth}}(\mathcal{A}) \\
&\leq n_\mathsf{E} \cdot n_\mathsf{N} \cdot n_\mathsf{J} \cdot \mathsf{adv}_2 + \mathsf{adv}_\Pi^{\mathsf{ent\text{-}auth}}(\mathcal{A}) \\
&\leq n_\mathsf{E} \cdot n_\mathsf{N} \cdot n_\mathsf{J} \left(\mathsf{adv}_3 + \mathsf{adv}_{P'}^{\mathsf{chan\text{-}sec}}(\mathcal{B}_1)\right) + \mathsf{adv}_\Pi^{\mathsf{ent\text{-}auth}}(\mathcal{A}) \\
&\leq n_\mathsf{E} \cdot n_\mathsf{N} \cdot n_\mathsf{J} \left(\mathsf{adv}_4 + 2\mathsf{adv}_{P'}^{\mathsf{chan\text{-}sec}}(\mathcal{B}_1)\right) + \mathsf{adv}_\Pi^{\mathsf{ent\text{-}auth}}(\mathcal{A}) \\
&\leq n_\mathsf{E} \cdot n_\mathsf{N} \cdot n_\mathsf{J} \left(\mathsf{adv}_5 + 3\mathsf{adv}_{P'}^{\mathsf{chan\text{-}sec}}(\mathcal{B}_1)\right) + \mathsf{adv}_\Pi^{\mathsf{ent\text{-}auth}}(\mathcal{A}) \\
&\leq n_\mathsf{E} \cdot n_\mathsf{N} \cdot n_\mathsf{J} \left(\mathsf{adv}_P^{\mathsf{chan\text{-}sec}}(\mathcal{B}_0) + 3\mathsf{adv}_{P'}^{\mathsf{chan\text{-}sec}}(\mathcal{B}_1)\right) + \mathsf{adv}_\Pi^{\mathsf{ent\text{-}auth}}(\mathcal{A})
\end{aligned}$$

□

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

In this section, we give the full proof of Theorem 5.2.

Entity Authentication. First we consider the entity authentication property.

Proof. Let $\mathsf{adv}_P^{\mathsf{ent\text{-}auth}}(\mathcal{A})$ be the probability that the adversary $\mathcal{A}$ wins the entity authentication game. Let $\mathsf{adv}_{P,\mathsf{client}}^{\mathsf{ent\text{-}auth}}(\mathcal{A})$ bounds the probability that a client (ED) adversary succeeds, and $\mathsf{adv}_{P,\mathsf{server}}^{\mathsf{ent\text{-}auth}}(\mathcal{A})$ bounds the probability that a server (NS-JS) adversary succeeds. We have that $\mathsf{adv}_P^{\mathsf{ent\text{-}auth}}(\mathcal{A}) \leq \mathsf{adv}_{P,\mathsf{client}}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) + \mathsf{adv}_{P,\mathsf{server}}^{\mathsf{ent\text{-}auth}}(\mathcal{A})$.

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

$$\Pr[E_0] = \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P,\mathsf{client}}(\mathcal{A}).$$

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 $\mathcal{B}_1$ able to distinguish the KDF_{mk} function from a random function. Hence

$$\Pr[E_1] - \Pr[E_2] \leq \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}_{mk}}(\mathcal{B}_1) = \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1).$$

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 $\mathcal{B}_0$ able to distinguish the MAC function from a random function. Hence

$$\Pr[E_2] - \Pr[E_3] \leq \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0).$$

Game 4. The server instance computes the Join Accept message in the following way: 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 $\mathcal{B}_2$ able to distinguish $\mathsf{AES}^{-1}(MK_1, \cdot)$ from a random permutation. Hence

$$\Pr[E_3] - \Pr[E_4] \leq \mathsf{adv}^{\mathsf{prp}}_{\mathsf{AES}}(\mathcal{B}_2).$$

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 verifies τ_J and checks that cnt_J is correct. 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 $\mathcal{B}_1$ able to distinguish the KDF_a function from a random function. Hence

$$\Pr[E_5] - \Pr[E_6] \leq \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}_a}(\mathcal{B}_1) = \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1).$$

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^{i2}, and (optionally) K_a^e are used to compute the RekeyConf message. K_c^e, K_c^{i2} 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^{i2}, 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

$$\Pr[E_6] - \Pr[E_7] \leq \mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{server}}}(\mathcal{B}_3).$$

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

$$\begin{aligned}
\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P,\mathsf{client}}(\mathcal{A}) &= \Pr[E_0] \\
&= q \cdot n_{\mathsf{C}} \cdot \Pr[E_1] \\
&\leq q \cdot n_{\mathsf{C}} \Big(\Pr[E_2] + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) \Big) \\
&\leq q \cdot n_{\mathsf{C}} \Big(\Pr[E_3] + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) \Big) \\
&\leq q \cdot n_{\mathsf{C}} \Big(\Pr[E_4] + \mathsf{adv}^{\mathsf{prp}}_{\mathsf{AES}}(\mathcal{B}_2) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) \Big) \\
&\leq q \cdot n_{\mathsf{C}} \Big(\Pr[E_5] + 2^{-\mu}(1 - 2^{-\beta}) + \mathsf{adv}^{\mathsf{prp}}_{\mathsf{AES}}(\mathcal{B}_2) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) \\
&\qquad + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) \Big) \\
&\leq q \cdot n_{\mathsf{C}} \Big(\Pr[E_6] + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) + 2^{-\mu}(1 - 2^{-\beta}) + \mathsf{adv}^{\mathsf{prp}}_{\mathsf{AES}}(\mathcal{B}_2) \\
&\qquad + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) \Big) \\
&\leq q \cdot n_{\mathsf{C}} \big(\Pr[E_7] + \mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{server}}}(\mathcal{B}_3) + 2^{-\mu}(1 - 2^{-\beta}) + \mathsf{adv}^{\mathsf{prp}}_{\mathsf{AES}}(\mathcal{B}_2) \\
&\qquad + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) \Big) \\
&\leq q \cdot n_{\mathsf{C}} \Big(\mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{server}}}(\mathcal{B}_3) + 2^{-\mu}(1 - 2^{-\beta}) + \mathsf{adv}^{\mathsf{prp}}_{\mathsf{AES}}(\mathcal{B}_2) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) \\
&\qquad + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) \Big)
\end{aligned}$$

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

$$\Pr[E_0] = \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P,\mathsf{server}}(\mathcal{A}).$$

Game 1. In this game, the challenger tries to guess which server instance π_j^t 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. We replace the MAC function used by the server instance π_j^t 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_j^t 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

$$\Pr[E_1] - \Pr[E_2] \leq \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0).$$

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. 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

$$\Pr[E_3] - \Pr[E_4] \leq \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}_c}(\mathcal{B}_1) = \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1).$$

Game 5. 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

$$\Pr[E_4] - \Pr[E_5] \leq \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}_a}(\mathcal{B}_1) = \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1).$$

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

$$\Pr[E_5] - \Pr[E_6] \leq \mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{client}}}(\mathcal{B}_3).$$

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

$$\begin{aligned}
\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P,\mathsf{server}}(\mathcal{A}) &= \Pr[E_0] \\
&= q \cdot n_{\mathsf{S}} \cdot \Pr[E_1] \\
&\leq q \cdot n_{\mathsf{S}} \left(\Pr[E_2] + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0)\right) \\
&\leq q \cdot n_{\mathsf{S}} \left(\Pr[E_3] + 2^{-\mu} + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0)\right) \\
&\leq q \cdot n_{\mathsf{S}} \left(\Pr[E_4] + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) + 2^{-\mu} + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0)\right) \\
&\leq q \cdot n_{\mathsf{S}} \left(\Pr[E_5] + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) + 2^{-\mu} + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0)\right) \\
&\leq q \cdot n_{\mathsf{S}} \left(\Pr[E_6] + \mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{client}}}(\mathcal{B}_3) + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) + 2^{-\mu} + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0)\right) \\
&\leq q \cdot n_{\mathsf{S}} \left(\mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{client}}}(\mathcal{B}_3) + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) + 2^{-\mu} + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0)\right)
\end{aligned}$$

Therefore we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P}(\mathcal{A}) &\leq \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P,\mathsf{client}}(\mathcal{A}) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P,\mathsf{server}}(\mathcal{A}) \\
&\leq q \cdot n_{\mathsf{C}} \Big(\mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{server}}}(\mathcal{B}_3) + 2^{-\mu}(1-2^{-\beta}) + \mathsf{adv}^{\mathsf{prp}}_{\mathsf{AES}}(\mathcal{B}_2) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) \\
&\qquad\qquad + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1)\Big) \\
&\quad + q \cdot n_{\mathsf{S}} \left(\mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{client}}}(\mathcal{B}_3) + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) + 2^{-\mu} + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0)\right) \\
&\leq q \Big[n_{\mathsf{S}} \cdot \mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{client}}}(\mathcal{B}_3) + n_{\mathsf{C}} \cdot \mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{server}}}(\mathcal{B}_3) \\
&\qquad + (n_{\mathsf{C}} + n_{\mathsf{S}}) \left(\mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1)\right) \\
&\qquad + n_{\mathsf{C}} \cdot \mathsf{adv}^{\mathsf{prp}}_{\mathsf{AES}}(\mathcal{B}_2) + 2^{-\mu}\left(n_{\mathsf{C}}(1-2^{-\beta}) + n_{\mathsf{S}}\right)\Big]
\end{aligned}$$

In addition, we have also

$$\begin{aligned}
\Pr[\text{forgery Join Request}] &\leq p_{jr} = \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) + 2^{-\mu} \\
\Pr[\text{forgery Join Accept}] &\leq p_{ja} = \mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{MAC}}(\mathcal{B}_0) + \mathsf{adv}^{\mathsf{prp}}_{\mathsf{AES}}(\mathcal{B}_2) + 2^{-\mu}(1-2^{-\beta})
\end{aligned}$$

□

Channel Security. Now we consider the channel security property.

Proof. Let $\mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P}(\mathcal{A})$ be the advantage of the adversary $\mathcal{A}$ in winning the channel security experiment against an instance of some client or server party. That is $\mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P}(\mathcal{A}) = \left|\Pr[\pi^s_i.\mathsf{b} = b] - \frac{1}{2}\right|$, 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

$$\Pr[E_0] = \frac{1}{2} + \mathsf{adv}^{\mathsf{chan\text{-}sec}}_{P}(\mathcal{A}) = \frac{1}{2} + \mathsf{adv}_0.$$

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

$$\mathsf{adv}_0 \leq \mathsf{adv}_1 + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P}(\mathcal{A}).$$

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 it 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 it targets. That is it 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

$$\mathsf{adv}_2 \leq \mathsf{adv}_3 + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}_c}(\mathcal{B}_1) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}_a}(\mathcal{B}_1) = \mathsf{adv}_3 + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{AES}}(\mathcal{B}_1).$$

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 $\mathsf{Encrypt}(\pi_i^s, \cdot)$ query to $\mathsf{Encrypt}(\cdot)$, and sends the response to $\mathcal{A}_{\mathsf{client}}$ (resp. $\mathcal{A}_{\mathsf{server}}$). It forwards any $\mathsf{Decrypt}(\pi_j^t, \cdot)$ query to $\mathsf{Decrypt}(\cdot)$, 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 $\mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{client}}}(\mathcal{B}_3)$ (resp. $\mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{server}}}(\mathcal{B}_3)$). Hence

$$\mathsf{adv}_4 \leq \mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{client}}}(\mathcal{B}_3) + \mathsf{adv}^{\mathsf{sae}}_{\mathsf{StAE}_{\mathsf{server}}}(\mathcal{B}_3).$$

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

$$\begin{aligned}
\mathsf{adv}_P^{\text{chan-sec}}(\mathcal{A}) &= \mathsf{adv}_0 \\
&\leq \mathsf{adv}_1 + \mathsf{adv}_P^{\text{ent-auth}}(\mathcal{A}) \\
&\leq q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}} \cdot \mathsf{adv}_2 + \mathsf{adv}_P^{\text{ent-auth}}(\mathcal{A}) \\
&\leq q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}} \left(\mathsf{adv}_3 + 2\mathsf{adv}_{\mathsf{AES}}^{\text{prf}}(\mathcal{B}_1)\right) + \mathsf{adv}_P^{\text{ent-auth}}(\mathcal{A}) \\
&\leq q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}} \left(\mathsf{adv}_4 + 2\mathsf{adv}_{\mathsf{AES}}^{\text{prf}}(\mathcal{B}_1)\right) + \mathsf{adv}_P^{\text{ent-auth}}(\mathcal{A}) \\
&\leq q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}} \left(\mathsf{adv}_{\mathsf{StAE}_{\mathsf{client}}}^{\text{sae}}(\mathcal{B}_3) + \mathsf{adv}_{\mathsf{StAE}_{\mathsf{server}}}^{\text{sae}}(\mathcal{B}_3) + 2\mathsf{adv}_{\mathsf{AES}}^{\text{prf}}(\mathcal{B}_1)\right) \\
&\quad + \mathsf{adv}_P^{\text{ent-auth}}(\mathcal{A})
\end{aligned}$$

□

5.4.3.3 sAE Security in LoRaWAN 1.1

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

Sketch of proof. Bellare and Namprempre [BN00] 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-secure. Moreover, Rogaway and Shrimpton [RS06] 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, Desai, Jokipii, and Rogaway [BDJR97] under the assumption that the block cipher is a good PRF. Iwata and Kurosawa [IK03c] show that CMAC is SUF-CMA-secure 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 $\mathsf{StAE}_{\mathsf{server}}$. □

SAKE: a Two-party AKE 6

Key exchange protocols in the asymmetric-key setting are known to provide stronger security properties than protocols in symmetric-key cryptography, as illustrated in Chapter 1. In particular, they can provide forward secrecy, as illustrated by protocols based on the Diffie-Hellman scheme. However public-key algorithms are too heavy for some class of low-resource devices, which can then not benefit from forward secrecy.

After the cryptanalysis of two widely deployed protocols in Chapters 3, and 4, we describe, in this chapter, a two-party scheme that aims at tackling the issue of forward secrecy in the symmetric-key setting. The authenticated key exchange protocol we present is solely based on symmetric-key functions, and yet does guarantee forward secrecy. We call it *Symmetric-key Authenticated Key Exchange*, or SAKE for short.

In addition, we describe SAKE-AM, a complementary mode of SAKE, that allows inverting the roles between the initiator and the responder. This yields an implementation such that any party can be either initiator or responder of a protocol run, while the smallest amount of calculation is always done by the same party (which, in practice, is the most constrained). This is particularly convenient in the context of IoT where a set of (low-resource) end-devices communicates with a back-end server (which has heavier computational capabilities).

Using a provable security approach, we show that the protocols are sound and secure in a strong security model (i.e., commonly used in the public-key setting).

Finally we investigate the possibility to devise a variant of SAKE which makes use of zero pseudo-random values.

The results of this chapter have been partially published in [ACF20].

Contents

6.1 Motivation

6.1.1 Context

An authenticated key exchange (AKE) protocol executed between two parties aims at providing unilateral or mutual entity authentication, and computing a fresh shared session key. Well-known two-party authenticated key exchange protocols make use of digital signatures to provide authentication, and apply the Diffie-Hellman (DH) scheme [DH76] to compute a shared session key. However, such protocols can be too heavy for low-resource devices. More suited protocols, solely based on symmetric-key functions, have been proposed (e.g., [BR94; PST+02; PS04; Int08; BM03b; Glo18; Zig14; Sor17] to cite a few), including widely deployed ones (e.g., in 3G/UMTS [3rda] and 4G/LTE [3rdb]). Such symmetric-key protocols are needed in various applications, ranging from Wireless Sensor Networks (WSNs), Radio Frequency Identification (RFID) tags, smart cards, Controller Area Networks (CANs) for vehicular systems, smart home, up to industrial Internet of Things (IoT). Yet, existing symmetric-key based protocols lack a fundamental security property usually provided by the DH scheme: *forward secrecy* [Gün90; DvW92].

Forward secrecy is a very strong form of long-term security which, informally, guarantees that future disclosures of some long-term secret keys do not compromise past session keys. It is widely accepted that this property can only be provided by asymmetric schemes (at least regarding stateless protocols). Indeed, in protocols based on symmetric-key functions, the two parties must share a long-term symmetric key (which the session keys are computed from). Therefore the disclosure of this static long-term key allows an adversary to compute all the past (and future) session keys (if the adversary has eavesdropped on the communications, as it is assumed in usual security models).

6.1.2 Forward Secrecy in the Symmetric-key Setting

In this section, we summarise several works that aim at achieving some form of forward secrecy in the symmetric-key setting. We stress that the goals of the schemes briefly described below are not necessarily the same as ours (essentially, performing a key exchange in our case). Nonetheless, the small number of existing symmetric-key protocols that provide forward secrecy, and the lukewarm security level they achieve illustrate that combining symmetric-key cryptography and (a strong form of) forward secrecy is a non-trivial task.

Dousti and Jalili [DJ14] describe a key exchange protocol where the shared master key is updated based on time. Their protocol requires perfect synchronism between the parties otherwise this leads to two main consequences. Firstly, in order to handle the key exchange messages, the parties may use different values of the master key corresponding to consecutive epochs, which causes the session to abort. Secondly, this allows an adversary to trivially break forward secrecy. Once a party deems the protocol run is correct and the session key can be safely used (i.e., once the party “accepts"), the adversary corrupts its partner (which still owns the previous, not updated yet, master key), and computes the current session key. Furthermore, achieving perfect time synchronisation may be quite complex in any context, in particular for low-resource devices. Contrary to Dousti and Jalili, the protocol we propose explicitly deals with the issue of updating the master keys at both parties without requiring any additional functionality (such as a synchronised clock).

In the RFID field, the protocol proposed by Le, Burmester, and de Medeiros [LBM07] aims at

authenticating a tag to a server, and at computing a session key in order to establish a secure channel (which they do not describe). The master key is updated throughout the protocol run. To deal with the possible desynchronisation between the reader and the tag, the server keeps two consecutive values of the key: the current and the previous one. If the tag does not update its master key (which happens when the last message is dropped), the server is able to catch up during the next session. This implies that, in case of desynchronisation, the server computes the session key from the updated master key, whereas the tag still stores the previous value. Hence, an adversary who corrupts the tag can compute the previous session key with respect to the server. In fact, since the server always keeps the previous value of the master key, together with the current one, the scheme is intrinsically insecure in strong security models (i.e., models that allow the adversary to corrupt any of the partners, once the targeted party accepts). Yet, Le et al. analyse their protocol in a model where any server corruption is forbidden, and corrupting a tag is allowed only once it accepts. In our scheme, one of the parties also keeps in memory (a few) samples of a master key corresponding to different epochs (including a previous one). Yet the disclosure of all these values does *not* compromise past session keys. Furthermore, the (strong) security model we use allows the adversary to corrupt either party as soon as the targeted party accepts.

Brier and Peyrin [BP10] propose a forward secret key derivation scheme in a client-server setting, that aims at improving a previous proposal [Ame09]. In addition to forward secrecy, another constraint is that the amount of calculation to compute the master key (directly used as encryption key) on the server side must be low. Their solution implies the storage, on the client side, of several keys in parallel and to use a (short) counter, which is involved in the keys update. The keys belong to a tree whose each leaf (key) is derived from the previous one and the counter. The client must send the counter with the encrypted message for the server to be able to compute the corresponding key. The main drawback of this scheme is that the number of possible encryption keys is reduced. Increasing that limit implies increasing the counter size and the number of keys stored in parallel on the client side. Moreover, Brier and Peyrin (as well as [Ame09]) focus on forward secrecy with respect to the client only. The server is deemed as incorruptible, and is supposed to compute an encryption key only upon reception of a client's message (the secure channel is unidirectional, and the server does not need to send encrypted messages to the client). Therefore, the scheme does not need to deal with the issue of *both* parties being in sync (with respect to the key computation), and providing forward secrecy. In addition, the purpose of Brier and Peyrin (as well as [Ame09]) is not to provide mutual authentication. More generally sending additional information in order to resynchronise (such as a sufficiently large counter) is a simple (and inefficient) way to build a forward secret protocol. But this yields several drawbacks. Firstly, the size of such a counter must be large enough in order to avoid any exhaustion. Secondly, sending the counter (at least periodically) is necessary for the two parties to resynchronise, which consumes bandwidth. Thirdly, resynchronisation may imply multiple updates of the master keys at once (the scheme of Brier and Peyrin and [Ame09] aims at limiting that amount of calculation, but it leads to a limited number of possible encryption keys). Our scheme avoids all these drawbacks.

The more general question of forward security in symmetric cryptography has been also investigated by Bellare and Yee [BY03]. They propose formal definitions and practical constructions of forward secure primitives (e.g., MAC, symmetric encryption algorithm). Their constructions protect against decryption of past messages, or antedated forgeries of messages (i.e., previously authenticated messages are made untrustworthy). Their algorithms are based on key-evolving

schemes [BM99]. Nonetheless, Bellare and Yee consider only algorithms (but not protocols) and they do not deal with the issue of synchronising the evolution of the shared key at *both* parties. That is, they propose out-of-context (non-interactive) solutions with respect to our purpose.

Abdalla and Bellare [AB00] investigate a related question which is "re-keying". Their formal analysis shows that appropriate re-keying techniques "increase" the lifetime of a key. They consider re-keying in the context of symmetric encryption (in order to thwart attacks based on the ability to get lots of encrypted messages under the same key), and forward security (in order to protect past keys). Yet, they confine their analysis to algorithms and not protocols. Hence, as Abdalla and Bellare [BY03], they do not treat the synchronisation issues that arise from evolving a shared symmetric key.

The Signal messaging protocol [Sig], devised by Marlinspike and Perrin, uses a key derivation scheme called "double ratchet algorithm" [PM16]. This scheme combines a DH based mechanism with a symmetric key-evolving mechanism (based on a one-way function). The first mechanism provides an asymmetric ratchet, whereas the second provides a symmetric ratchet. The asymmetric ratchet is applied when a fresh DH share is received (included in an application message) from the peer. The symmetric ratchet is applied when a party wants to send several successive messages without new incoming message from its partner. Thanks to the DH scheme, the asymmetric ratchet is supposed to provide forward secrecy.[1] Regarding the symmetric ratchet, each party is compelled to store the decryption keys of the not yet received messages. This is due to the asynchronous nature of the Signal protocol. Therefore, the symmetric ratchet in Signal does not provide forward secrecy, as stated in their security analysis by Cohn-Gordon, Cremers, Dowling, Garratt, and Stebila [CGCD+17]: "*old but unused receiving keys are stored at the peer for an implementation dependent length of time, trading off forward security for transparent handling of outdated messages. This of course weakens the forward secrecy of the keys*". Consequently, Cohn-Gordon et al. choose not to model this weakened property. In turn, Alwen, Coretti, and Dodis [ACD19] incorporate the latter in the security analysis of their "generalised Signal protocol". But the crucial difference in their notion of forward security is that, as soon as the receiver is compromised, no more security can be provided. On the contrary, we tackle the synchronisation issue, and solve it in our protocol. The security model we use captures forward secrecy and allows corrupting a party and its partner as soon as the targeted party "accepts" (i.e., deems the session key can be safely used). With regard to Signal, our protocol can be compared to the asymmetric ratchet (in synchronous mode), and yet does not implement asymmetric functions.

Table 6.1 provides a comparison between the aforementioned schemes with respect to six features:

- "2P" – The scheme is a two-party protocol (in opposition to a cryptographic primitive).
- "Bilateral" – The forward secrecy property is guaranteed at both parties (e.g., in contrast to a client-server context where the server is deemed as incorruptible).
- "Sym." – The scheme is built on symmetric-key functions only.
- "I/R" – Any party can be initiator or responder in a session.

[1] In Signal, the DH exchanges can be asynchronous. This impairs the forward secrecy property usually ensured by this scheme.

- "No +func." – No additional functionality is required in order to guarantee forward secrecy (e.g., synchronised clock).
- "Unlimited" – The number of sessions the scheme can execute, or the number of (encryption) keys it can output is (virtually) unlimited.

Table 6.1 – Comparison between several schemes aiming at ensuring forward secrecy

Feature / Scheme	2P	Bilateral	Sym.	I/R	No +func.	Unlimited
Dousti and Jalili [DJ14]	✓	✓	✓	✓	✗	✓
Le et al. [LBM07]	✓	✗	✓	✗	✓	✓
Brier and Peyrin [BP10]	✓	✗	✓	✗	✓	✗
Bellare and Yee [BY03]	✗	-	✓	-	-	-
Abdalla and Bellare [AB00]	✗	-	✓	-	-	✓
Signal [Sig]	✓	✗	✗	✓	✓	✓
Our SAKE/SAKE-AM protocol	✓	✓	✓	✓	✓	✓

6.2 Symmetric-key AKE Protocol with Forward Secrecy

6.2.1 In a Nutshell

Our *Symmetric-key Authenticated Key Exchange* (SAKE) protocol provides mutual authentication and key agreement, and guarantees also forward secrecy. We attain this very strong form of long-term security by using a key-evolving scheme. As soon as two parties make a shared (symmetric) key evolve, a synchronisation problem arises. We provide a shrewd solution to this issue. We require using neither a clock, nor an additional resynchronising procedure. Our solution is based on a second (independent) chain of master keys. These keys allow tracking the evolution of the internal state, and resynchronising the parties if necessary. The parties authenticate each other prior to updating their master keys. Hence the possible gap is bounded (as we prove it), and each party is always able to catch up in case of desynchronisation (of course, if the session is correct and complete). Mutual authentication, key exchange (with forward secrecy), and resynchronisation are done in the continuity of the protocol run. In addition, the protocol we describe has the following characteristics.

- It is self-synchronising. That is, after a correct and complete session (and whatever the internal state of the parties prior to the session), the two parties involved in the protocol run share a new session key, and their internal state is updated and synchronised.
- It allows establishing an (virtually) unlimited number of sessions (as opposite to protocols that make use of a predefined list of master keys, each being used once only).
- The amount of calculation done by both parties in a single protocol run is strictly bounded. In particular we avoid the need of sending additional information in order to resynchronise, such as a (sufficiently large) counter that keeps track of the evolution of the master keys, and the subsequent drawbacks: periodically doing a great amount of computations at once (when resynchronisation is necessary), and consuming bandwidth (to transmit the additional data).

6.2.2 Key Concepts

The protocol allows two parties A (initiator) and B (responder) to mutually authenticate and compute a shared session key. It is based on two types of master keys: a derivation master key K (used to compute the session keys), and an authentication master key K' (used to authenticate the messages sent during a protocol run). The protocol makes use of symmetric-key functions only. Each pair of parties (A, B) shares distinct master keys. The main lines of the protocol are as follows: the two parties exchange pseudo-random values r_A, r_B. These two values are used to

- authenticate each other: each party sends back the value it has received in a message that is MAC-ed with the authentication master key K'. For instance, if B receives r_A it replies with $r_B \| \tau_B$ where $\tau_B = \mathsf{MAC}(K', B \| A \| r_B \| r_A)$.
- Compute a session key: a pseudo-random function KDF is keyed with the derivation master key K and uses the pseudo-random values as input. That is, $sk \leftarrow \mathsf{KDF}(K, f(r_A, r_B))$. Function f is deliberately left undefined. For instance, $f(r_A, r_B)$ can be equal to the concatenation or the bitwise addition of r_A and r_B.[2]

Providing forward secrecy. The shared key K is used to compute the session keys. If this key remains unchanged throughout all sessions, its disclosure allows computing all past (and future) session keys. To solve this issue we apply a key-evolving technique. We update the master key such that a previous version of the latter cannot be computed from an updated one. Each of the two parties involved in a session updates its own copy of the derivation master key K with a non-invertible function update: $K \leftarrow \mathsf{update}(K)$. Hence this protects past sessions in case the (current value of) master key K is revealed. Each party authenticates its peer prior to updating the derivation master key. If the master key is updated throughout the session, it may happen that one of the two involved parties update its master key whereas the other does not. This leads to a *synchronisation problem.*

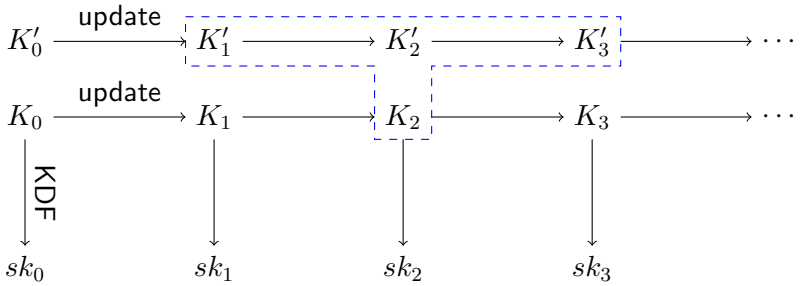


Figure 6.1 – Master key chains in SAKE. At epoch j, the initiator stores four keys: $K = K_j$, and K'_{j-1}, K'_j, K'_{j+1}. The responder stores two keys: $K = K_j$ and $K' = K'_j$. An illustration with $j = 2$ corresponds to the keys surrounded by the blue dashed box.

The synchronisation problem. If two parties use a different key K, they are obviously not able to compute a shared session key. Hence they must resynchronise first. More fundamentally,

[2]The function f must be chosen such that the security of KDF is not impaired. We assume here that the cryptographic functions used are ideal (investigating this topic is beyond the scope of this chapter).

if a party initiates a session with some derivation master key K, and its partner stores a master key corresponding to an earlier epoch, then an adversary who corrupts the partner can compute past session keys with respect to the initiator, hence trivially break forward secrecy. Therefore, it is of paramount importance that the parties know if the master key of its partner has actually been updated. We provide a solution to both issues in the continuity of a *single* session. In particular, no extra procedure is needed, and no (heavy) additional data is sent in order for a desynchronised party to catch up.

We base our solution on the second master key K' used to authenticate the messages exchanged during a session. The solution is to update K' at the same time as K. Therefore the evolution of K' follows that of K. The party that receives the first authenticated message uses the MAC tag to learn which epoch the sender belongs to. Of course, K' can also be desynchronised in the same way as K. This is why, whereas one party (responder B) stores only one sample of the key K', the other party (initiator A) stores several samples of the authentication master key K' corresponding to several consecutive epochs. We prove (in Section 6.3.1) that only three keys K'_{j+1}, K'_j, K'_{j-1}, corresponding respectively to the next, the current, and the previous epochs, are sufficient in order for A and B to resynchronise. The initiator (A) is the one able to deal with the synchronisation issue, and consequently tells B how to behave. Each party "accepts" only after it has received a confirmation (final MAC-ed messages) that its partner has already updated its own master keys. In such a case, the party ending in accepting state deems that the fresh session key can be safely used. Otherwise (in particular when the parties are desynchronised), the session key is discarded.

Since two independent master keys are used (authentication and session key derivation), one can safely maintain a copy of K' corresponding to an earlier epoch (K'_{j-1}) without risk of threatening forward secrecy. Only one sample of the derivation master key K is kept: the most up-to-date.

6.2.3 Description of the Protocol

Our SAKE protocol is depicted by Figure 6.2. The parameter δ_{AB} computed by A corresponds to the gap between A and B with respect to the evolution of the master keys. We prove that $\delta_{AB} \in \{-1, 0, 1\}$ (see Section 6.3.1). That is, A can only be either *one step* behind, or in sync, or *one step* ahead to B. During a session, A uses the keys K'_j, K'_{j-1}, K'_{j+1} (by order of likelihood) and the first message (m_B) sent by B to learn δ_{AB}. The message m_B is computed with the current value K' of B. Therefore m_B indicates the current synchronisation state of B. Then A informs B. One bit ϵ is enough (message m_A) because B takes two behaviours only: if $\delta_{AB} \in \{-1, 0\}$ ($\epsilon = 0$), and if $\delta_{AB} = 1$ ($\epsilon = 1$). A and B behave as follows.

- If A is in sync with B ($\delta_{AB} = 0$), A computes the new session key, and updates its master keys. Then, upon reception of m_A, B does the same.
- If A is in advance ($\delta_{AB} = 1$), A waits for B to resynchronise (i.e., B updates its master keys a first time), and to proceed with the regular operations (i.e., B computes the new session key, and updates its master keys a second time). Then, once A receives a confirmation that B is synchronised (message τ'_B), A performs the regular operations as well (session key computation, master keys update). Since A waits for B to resynchronise before proceeding, the gap between the parties is *bounded* (as proved in Section 6.3.1).
- If A is late ($\delta_{AB} = -1$), it resynchronises (i.e., it updates its master keys a first time), and then performs the regular operations (session key computation, master keys update). Then (upon reception of message m_A), B applies the regular operations.

Once a correct and complete session ends, three goals are achieved in the *same* protocol run: (i) the two parties have updated their master keys, (ii) they are synchronised (which stems in particular from the fact that the gap between A and B is bounded, i.e., $|\delta_{AB}| \leq 1$), and (iii) they share a new session key. In other words, the protocol is *self-synchronising*.

The session can be reduced from five to four messages in some cases. Indeed, regarding the synchronisation state, in two cases (when $\delta_{AB} \in \{-1, 0\}$, that is $\epsilon = 0$), A and B are synchronised, and share a session key once B has received message m_A and executed the subsequent operations. Therefore, in such a case, the session can end upon reception of message τ'_B by A. More precisely

- if $\delta_{AB} = 1$ ($\epsilon = 1$), then A accepts upon reception of τ'_B, and B accepts upon reception of τ'_A;
- if $\delta_{AB} \in \{-1, 0\}$ ($\epsilon = 0$), then A accepts upon reception of τ'_B, and B accepts upon reception of m_A.

Although this does not appear explicitly in Figure 6.2, a party aborts the session if it receives a message computed with an invalid identity. For the responder B, an invalid identity corresponds to an initiator party A it does not share master keys with. For an initiator A, the particular case $B = A$, among other possibilities, yields an error (i.e., each party must have a distinct identity).

Note that, since K'_{j+1} and K'_j can be computed from K'_{j-1}, it is also possible to store only K'_{j-1}, and to compute the two other keys when necessary during the session.

Remark. With respect to the security model presented in Chapter 2, Section 2.3.1, the long-term key of A and B corresponds respectively to $A.\mathsf{ltk} = (K, K'_{j-1})$ and $B.\mathsf{ltk} = (K, K')$. We could have allowed the authentication master key K'_{j-1}/K' to be disclosed prior to the start of the session. This would not impair the forward secrecy of the derivation master key K. Nonetheless, knowing the authentication master key an adversary could desynchronise a legitimate party so that the party could not catch up anymore. Hence our choice to include both master keys in the response to a Corrupt-query.

A variant. Alternatively, the evolving authentication keys K' and K'_{j-1}, K'_j, K'_{j+1} can be replaced by a static authentication master key K', and two local counters c_A, c_B (respectively stored by A and B) that keep track of the evolution of the derivation master key K.[3] On the initiator's side, the MAC verifications are then done with consecutive values of the counter $j - 1, j, j + 1$.

Overall, the sequence of operations and the computations are similar to that of SAKE. This means mainly replacing function $x \mapsto \mathsf{MAC}(K'_j, x)$ with $x \mapsto \mathsf{MAC}(K', j\|x)$. This alternative implies the storage of two keys and one counter: K, K' and c_A/c_B, instead of two keys only: K and K'_{j-1}/K' (and, on the initiator's side only, one or two additional calls to update in order to compute K'_j and, possibly, K'_{j+1}).

Notation. For the sake of clarity, we use the following notation in Figure 6.2:

- kdf corresponds to: $sk \leftarrow \mathsf{KDF}(K, f(r_A, r_B))$
- upd_A corresponds to

[3] This alternative has been suggested by anonymous reviewers from Crypto 2019.

A $(K, K'_{j+1}, K'_j, K'_{j-1})$		B (K, K')
$r_A \xleftarrow{\$} \{0,1\}^\lambda$	$\xrightarrow{A\|r_A}$	
		$r_B \xleftarrow{\$} \{0,1\}^\lambda$ $\tau_B \leftarrow \mathsf{MAC}(K', B\|A\|r_B\|r_A)$ $m_B \leftarrow r_B\|\tau_B$
	$\xleftarrow{m_B}$	
`if` $(\mathsf{Vrf}(K'_j, B\|A\|r_B\|r_A, \tau_B) = \mathtt{true})$ $\delta_{AB} \leftarrow 0$ $K' \leftarrow K'_j$; kdf; upd_A; $\epsilon \leftarrow 0$ `else if` $(\mathsf{Vrf}(K'_{j-1}, B\|A\|r_B\|r_A, \tau_B) = \mathtt{true})$ $\delta_{AB} \leftarrow 1$ $K' \leftarrow K'_{j-1}$; $\epsilon \leftarrow 1$ `else if` $(\mathsf{Vrf}(K'_{j+1}, B\|A\|r_B\|r_A, \tau_B) = \mathtt{true})$ $\delta_{AB} \leftarrow -1$ $K' \leftarrow K'_{j+1}$; upd_A; kdf; upd_A; $\epsilon \leftarrow 0$ `else` abort		
$\tau_A \leftarrow \mathsf{MAC}(K', \epsilon\|A\|B\|r_A\|r_B)$ $m_A \leftarrow \epsilon\|\tau_A$		
	$\xrightarrow{m_A}$	
		`if` $(\mathsf{Vrf}(K', \epsilon\|A\|B\|r_A\|r_B, \tau_A) = \mathtt{false})$ abort `if` $(\epsilon = 1)$ upd_B kdf; upd_B $\tau'_B \leftarrow \mathsf{MAC}(K', r_B\|r_A)$
	$\xleftarrow{\tau'_B}$	
`if` $(\epsilon = 0)$ $K' \leftarrow K'_j$ `if` $(\mathsf{Vrf}(K', r_B\|r_A, \tau'_B) = \mathtt{false})$ abort `else if` $(\epsilon = 1)$ $K' \leftarrow K'_{j+1}$ `if` $(\mathsf{Vrf}(K', r_B\|r_A, \tau'_B) = \mathtt{false})$ abort kdf; upd_A		
$\tau'_A \leftarrow \mathsf{MAC}(K', r_A\|r_B)$		
	$\xrightarrow{\tau'_A}$	
		`if` $(\mathsf{Vrf}(K', r_A\|r_B, \tau'_A) = \mathtt{false})$ abort

Figure 6.2 – SAKE protocol

1. $K \leftarrow \mathsf{update}(K)$
2. $K'_{j-1} \leftarrow K'_j$
3. $K'_j \leftarrow K'_{j+1}$
4. $K'_{j+1} \leftarrow \mathsf{update}(K'_{j+1})$

- upd_B corresponds to
  1. $K \leftarrow \mathsf{update}(K)$
  2. $K' \leftarrow \mathsf{update}(K')$

Moreover, $\mathsf{Vrf}(k, m, \tau)$ denotes the MAC verification function that takes as input a secret key k, a message m, and a tag τ. It outputs `true` if τ is a valid tag on message m with respect to k. Otherwise, it returns `false`.

Before the first session between A and B, the master keys are initialised as follows[4]:

- K and K' are uniformly chosen at random.
- $K'_{j-1} \leftarrow \perp$
- $K'_j \leftarrow K'$
- $K'_{j+1} \leftarrow \mathsf{update}(K')$

6.3 Proofs for SAKE

In this section we prove that (i) SAKE is *sound*, and (ii) it is a *secure AKE* protocol according to Definition 2.10.

6.3.1 Soundness of SAKE

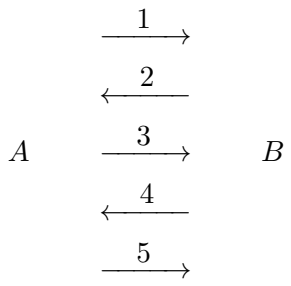
We want to show that SAKE is *sound*, which essentially means that, once a correct session is complete, both parties have updated their respective internal state, are synchronised, and share the same (new) session key. We call a "benign" adversary an adversary that faithfully forwards all messages between an initiator A and a responder B.

Lemma 6.1. *Let A and B be respectively the initiator and the responder of a SAKE session. Let δ_{AB} be the gap between A and B with respect to the evolution of the master keys of both parties. The following conditions always hold:*

1. *$\delta_{AB} \in \{-1, 0, 1\}$, and*
2. *whatever the synchronisation state between A and B at the beginning of a session (i.e., whatever A and B are synchronised or not), when that session completes in presence of a benign adversary, then*
   a) *A and B have updated their master keys at least once, and*
   b) *A and B are synchronised (with respect to their master keys), and*
   c) *A and B share the same session key.*

[4] During the first protocol run, A needs only K'_j to verify message m_B.

In order to prove Lemma 6.1, we use the following notation. The messages exchanged during a session are numbered in a natural way:



The notation "(i_A, i_B)" means that, when the session ends, the last valid message received by A is message of index i_A, and the last valid message received by B is message of index i_B. We call a (i_A, i_B)-*session* a session where the last message received by A is message i_A, and the last message received by B is message i_B. By convention $i_A = 0$ means that no message has been received by A.

It may happen that A send a first message which is not received by B. B cannot know if it has missed a first message. But this is of no consequence regarding the synchronisation between A and B (A may simply run the protocol anew). Therefore we do not use the value $i_B = 0$ (it is equivalent to $i_B = 5$). At initialisation (i.e., before the first run of the protocol), (i_A, i_B) is set to $(4, 5)$. Since A sends message $i \in \{3, 5\}$ only upon reception of a valid message $i - 1$, and B sends message $j \in \{2, 4\}$ only upon reception of a valid message $j - 1$, the only possible values for (i_A, i_B) are as listed in Table 6.2.

Table 6.2 – Possible values for (i_A, i_B) in SAKE

i_A \ i_B	1	3	5
0	✓	✗	✗
2	✓	✓	✗
4	✗	✓	✓

The diagram depicted by Figure 6.3 represents all the possible sequences of sessions with SAKE.

Proof. We prove Lemma 6.1. We first prove item 1.

Let c_A (resp. c_B) be a (virtual) monotonically increasing counter initialised to 0 that follows the evolution of the master keys held by A (resp. B). That is, c_A (resp. c_B) is increased each time the master keys K, K'_{j+1}, K'_j, K'_{j-1} (resp. K, K') are updated. The parameter δ_{AB} corresponds to the gap between A and B with respect to the evolution of their master keys, hence $\delta_{AB} = c_A - c_B$.

The different possible sessions are listed in Table 6.2. We prove item 1 by constructing iteratively Table 6.3a.

Before the first session, A and B are synchronised. That is $\delta_{AB} = c_A - c_B = 0$, and $(c_A, c_B) = (i, i)$ (with $i = 0$). Therefore, A can validate τ_B (in message m_B) with the same key $K'_j = K'$ as B. Hence A computes $\delta_{AB} = 0$, and $\epsilon = 0$. Consequently, if one carries out the

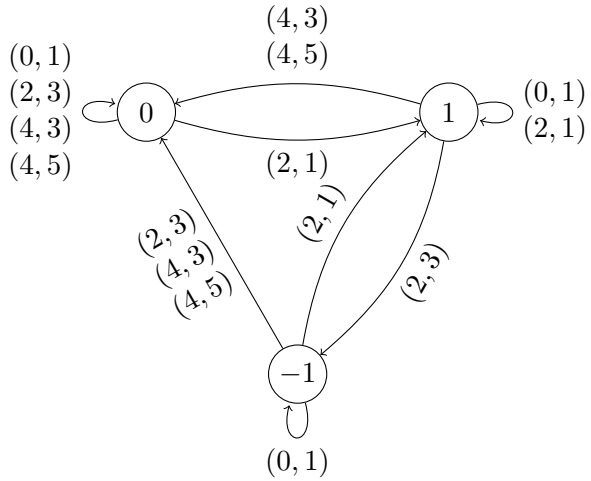


Figure 6.3 – Diagram of SAKE. The circled values correspond to the gap δ_{AB}, and each edge to a (i_A, i_B)-session.

protocol run starting with $\delta_{AB} = 0$ and $\epsilon = 0$, for each possible value (i_A, i_B), one eventually gets the following:

- $(c_A, c_B) = (i, i)$ and $\delta_{AB} = 0$ after a $(0, 1)$-session,
- $(c_A, c_B) = (i + 1, i)$ and $\delta_{AB} = 1$ after a $(2, 1)$-session,
- $(c_A, c_B) = (i + 1, i + 1)$ and $\delta_{AB} = 0$ after a $(2, 3)$-session,
- $(c_A, c_B) = (i + 1, i + 1)$ and $\delta_{AB} = 0$ after a $(4, 3)$-session,
- $(c_A, c_B) = (i + 1, i + 1)$ and $\delta_{AB} = 0$ after a $(4, 5)$-session.

This corresponds to the first column of Tables 6.3a and 6.3b. As we can see, the only possible values for δ_{AB} after any session are 0 and 1. $\delta_{AB} = 0$ has already been investigated. Hence, starting with $\delta_{AB} = 1$ (i.e., $(c_A, c_B) = (i + 1, i)$), we look for all the values δ_{AB} may have when the session ends, considering any possible session.

$(c_A, c_B) = (i + 1, i)$ means that A is in advance with respect to B. In such a case, A succeeds in validating τ_B with K'_{j-1} (and, indeed, finds $\delta_{AB} = 1$). Then A uses $\delta_{AB} = 1$ and $\epsilon = 1$. If one carries out the protocol run using these two values, one gets:

- $(c_A, c_B) = (i + 1, i)$ and $\delta_{AB} = 1$ after a $(0, 1)$-session,
- $(c_A, c_B) = (i + 1, i)$ and $\delta_{AB} = 1$ after a $(2, 1)$-session,
- $(c_A, c_B) = (i + 1, i + 2)$ and $\delta_{AB} = -1$ after a $(2, 3)$-session,
- $(c_A, c_B) = (i + 2, i + 2)$ and $\delta_{AB} = 0$ after a $(4, 3)$-session,
- $(c_A, c_B) = (i + 2, i + 2)$ and $\delta_{AB} = 0$ after a $(4, 5)$-session.

This corresponds to the second column of Table 6.3a. This shows that a third value is possible for δ_{AB}, which is -1 (i.e., $(c_A, c_B) = (i, i + 1)$).

Then we restart the protocol with all possible sessions, assuming that $(c_A, c_B) = (i, i + 1)$ at the beginning of the run. This means that A is one step late with respect to B. In such a

case, A succeeds in validating τ_B with key K'_{j+1} (and, indeed, finds $\delta_{AB} = -1$). Then A uses $\delta_{AB} = -1$ and $\epsilon = 0$. If one carries out the protocol run using these two values, one gets:

- $(c_A, c_B) = (i, i+1)$ and $\delta_{AB} = -1$ after a $(0,1)$-session,
- $(c_A, c_B) = (i+2, i+1)$ and $\delta_{AB} = 1$ after a $(2,1)$-session,
- $(c_A, c_B) = (i+2, i+2)$ and $\delta_{AB} = 0$ after a $(2,3)$-session,
- $(c_A, c_B) = (i+2, i+2)$ and $\delta_{AB} = 0$ after a $(4,3)$-session,
- $(c_A, c_B) = (i+2, i+2)$ and $\delta_{AB} = 0$ after a $(4,5)$-session.

We end with three possible values for δ_{AB} (third column of Table 6.3a): -1, 0 and 1, that have already been explored. This proves that, whatever the sequences of sessions, the only possible values for δ_{AB} are in $\{-1, 0, 1\}$.

Table 6.3 – Possible values for δ_{AB} and (c_A, c_B) among all sequences of sessions in SAKE

(a) Possible values for δ_{AB}

session \ δ_{AB}	0	1	-1
$(0,1)$	0	1	-1
$(2,1)$	1	1	1
$(2,3)$	0	-1	0
$(4,3)$	0	0	0
$(4,5)$	0	0	0

(b) Possible values for (c_A, c_B)

session \ (c_A, c_B)	(i,i)	$(i+1,i)$	$(i,i+1)$
$(0,1)$	(i,i)	$(i+1,i)$	$(i,i+1)$
$(2,1)$	$(i+1,i)$	$(i+1,i)$	$(i+2,i+1)$
$(2,3)$	$(i+1,i+1)$	$(i+1,i+2)$	$(i+2,i+2)$
$(4,3)$	$(i+1,i+1)$	$(i+2,i+2)$	$(i+2,i+2)$
$(4,5)$	$(i+1,i+1)$	$(i+2,i+2)$	$(i+2,i+2)$

Now we prove item 2 of Lemma 6.1.

We know that $\delta_{AB} \in \{-1, 0, 1\}$. For each possible value of δ_{AB} at the beginning of the session, the last line of Table 6.3a indicates the value of that parameter after a correct and complete session (i.e., a $(4,5)$-session). As we can see, A and B are always synchronised (i.e., $\delta_{AB} = 0$) in such a case whatever the value of δ_{AB} when the session starts. Furthermore, the session key computation immediately precedes the last update of the derivation master key K. Hence, when a correct and complete session ends, A and B use the same derivation master key K to compute the session key. Therefore, using the same values r_A, r_B, A and B compute the same session key.

In addition, Table 6.3b shows that, whatever the synchronisation state of A and B (i.e., c_A and c_B) at the beginning of the session, after a correct and complete session, A and B have updated their internal state at least once (as the last line of the table, corresponding to a $(4,5)$-session, indicates). □

6.3.2 Security of SAKE

In order to prove that the protocol SAKE is a secure AKE protocol, we use the AKE security model described in Chapter 2, Section 2.3.1. We define the partnering between two instances with the notion of *matching conversations* (see Definition 2.1). That is, 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 one.

We prohibit *parallel* executions of the protocol. Indeed, since the protocol we propose is based on shared evolving symmetric keys, running multiple instances in parallel may cause some executions to abort (we elaborate more on this in Section 6.6). This is the only restriction we demand compared to AKE model. In addition, for each party P_i we define the long-term key $P_i.\mathsf{ltk}$ to be $P_i.\mathsf{ltk} = (K, K'_{j-1})$ if $\rho = \mathsf{init}$, and $P_i.\mathsf{ltk} = (K, K')$ if $\rho = \mathsf{resp}$. The same long-term key ltk is shared by a unique pair of parties (P_i, P_j). That is, $P_i.\mathsf{ltk} = P_j.\mathsf{ltk}$.

Furthermore, we choose the function update to be a PRF, that is $\mathsf{update} : K \mapsto \mathsf{PRF}_{\mathsf{update}}(K, x)$ for some (constant) value x.

Theorem 6.2. *The protocol SAKE is a secure AKE protocol, and for any probabilistic polynomial time adversary $\mathcal{A}$ in the AKE security experiment against SAKE*

$$\begin{aligned}
\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE}(\mathcal{A}) &\leq nq\left((nq-1)2^{-\lambda} + (q+1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + 2\mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Tag}}(\mathcal{C})\right)\\
\mathsf{adv}^{\mathsf{key\text{-}ind}}_{SAKE}(\mathcal{A}) &\leq nq\left((q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}(\mathcal{D})\right) + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE}(\mathcal{A})
\end{aligned}$$

where n is the number of parties, q the number of instances (sessions) per party, λ the size of the pseudo-random values (r_A, r_B), and $\mathcal{B}$ is an adversary against the PRF-security of update*, $\mathcal{C}$ an adversary against the SUF-CMA-security of* $\mathsf{Tag} = (\mathsf{Tag.Gen}, \mathsf{Tag.MAC}, \mathsf{Tag.Vrf})$*, and $\mathcal{D}$ an adversary against the PRF-security of* KDF*.*

We give a proof of Theorem 6.2.

Entity authentication. First we consider the entity authentication experiment described in Chapter 2, Section 2.3.1.

Proof. Let E_i be the event that the adversary succeeds in making an instance accept maliciously in Game i. We use the following hops.

In order for an initiator instance π_i^s at some party P_i to accept, two valid messages (i.e., with valid MAC tags) must be received by π_i^s (m_B and τ'_B). We reduce the security of the Tag function to the (in)ability to forge a valid output. Therefore we use the fact that the key K' is random. By assumption, the genuine value of K' (i.e., the value used during the first session between two same parties) is uniformly chosen at random. Yet K' (and K) is updated throughout the session with the function update. If K' is random, we can rely on the pseudo-randomness of $\mathsf{update}(\cdot) = \mathsf{PRF}_{\mathsf{update}}(\cdot,\cdot)$. In turn, since $\mathsf{PRF}_{\mathsf{update}}(K', \cdot)$ can be replaced with a truly random function, its output (updated K') is random. Therefore, one can rely upon the pseudo-randomness of the function update keyed with this new value K', and so forth. Each transition (i.e., each update of K') implies a loss equal to $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B})$ corresponding to the ability of an adversary $\mathcal{B}$ to distinguish update from a random function.

If P_i is synchronised with the responder ($\delta_{AB} = 0$), P_i updates its master keys once (upon reception of m_B). If P_i is in advance ($\delta_{AB} = 1$), it updates its keys at most once (if a valid message τ'_B is received). If P_i is late ($\delta_{AB} = -1$), it updates its keys twice. Yet, in that case, P_i did not update its keys during the previous session. Therefore, on average, P_i updates its keys

at most once per session. Hence, when the u-th session starts, P_i has updated its keys at most $u-1$ times on average, and, upon reception of τ'_B, P_i updates the keys at most two times.

This is similar regarding the responder. A responder instance π_j^t at some party P_j accepts only if the two messages m_A and τ'_A are valid. Upon reception of a valid message m_A, the keys are updated once ($\epsilon = 0$) or twice ($\epsilon = 1$). In the latter case, the keys have not been updated during the previous session. This means that the keys are updated on average at most once per session. Therefore, when the u-th session starts, P_j has updated its keys at most $u-1$ times on average, and, upon reception of m_A, the keys are updated at most two times.

We can now proceed with the proof. We proceed through a sequence of games [Sho04; BR04], where each consecutive game aims at reducing the challenger's dependency on the functions Tag, update and KDF. Let E_i be the event that the adversary win the entity authentication experiment in Game i.

Game 0. This game corresponds to the entity authentication security experiment. Therefore

$$\Pr[E_0] = \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE}(\mathcal{A}).$$

Game 1. In this game, we add an abort rule. The challenger aborts if there exists any instance that chooses a random value r_A or r_B that is not unique. There is at most $n \times q$ random values, each uniformly drawn at random in $\{0,1\}^\lambda$. Therefore the probability that at least two random values be equal is at most $\frac{nq(nq-1)}{2^\lambda}$. Hence

$$\Pr[E_0] \leq \Pr[E_1] + \frac{nq(nq-1)}{2^\lambda}.$$

Game 2. In this game, we add an abort rule. The challenger tries to guess which instance will be the first to accept maliciously. If the guess is wrong, the game is aborted. The number of instances is at most nq. Therefore

$$\Pr[E_2] = \Pr[E_1] \times \frac{1}{nq}.$$

Game 3. Let π be the instance targeted by the adversary. In this game, we add an abort rule. The challenger aborts the experiment if π, behaving as an initiator (resp. responder) instance, ever receives a valid message m_B (resp. m_A) but no instance having a matching conversation to π has output that message. We reduce the probability of this event to the security of the functions Tag and update. As explained above, when the u-th session starts, the master keys have been updated at most $u-1$ times already. The genuine value of K' is uniformly chosen at random. In order to be able to replace, during the current session, the key used to compute the MAC tag in m_A (resp. m_B) with a random value, one must rely upon the pseudo-randomness of the function update that outputs (the new value of) K'. In turn, this relies upon the (previous) key K' being random (and on the pseudo-randomness of update). Therefore, in order to replace K' with a random value one must take into account the successive losses $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B})$, each corresponding to the ability of an adversary $\mathcal{B}$ to distinguish the function update (keyed with a different key K') from a random function. Since there is at most q sessions, this loss is at most $(q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B})$. Then we reduce the probability of the adversary $\mathcal{A}$ to win this game to the ability of an adversary $\mathcal{C}$ to forge a valid tag τ_B (resp. τ_A).

Therefore, we replace each function $\mathsf{update}(K') = \mathsf{PRF}_{\mathsf{update}}(K', x)$ (keyed with a different key K' throughout the, at most, $q-1$ successive sessions established, prior to that current session,

by the same party that owns π) with truly random functions $\mathsf{F}_0^{\mathsf{update}}, \ldots, \mathsf{F}_{q-2}^{\mathsf{update}}$. Moreover, if an instance uses the same key $K' = K'_i$, $0 \leq i < q-1$, to key update, then we replace update with the corresponding random function $\mathsf{F}_i^{\mathsf{update}}$. Since, to that point, the key $K' = K'_{q-1}$ used to compute the authentication tag τ_B (resp. τ_A) is random, we reduce the ability of $\mathcal{A}$ to win to the security of the Tag function. Hence

$$\Pr[E_2] \leq \Pr[E_3] + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + \mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Tag}}(\mathcal{C}).$$

Game 4. In this game, we add an abort rule. The challenger aborts the experiment if π ever receives a valid message τ'_B (resp. τ'_A), but no instance having a matching conversation to π has output that message. Between the message m_B (resp. m_A) being received by π, and the message τ'_B (resp. τ'_A) being received by π, the master keys are updated at most twice. We reduce the probability of the adversary to win this game to the security of the Tag function used to compute the message τ'_B (resp. τ'_A). In turn we must rely on the randomness of the MAC key, hence on the security of the function update used to update the MAC key K' (recall that, due to Game 3, the current key K' is random). Therefore

$$\Pr[E_3] \leq \Pr[E_4] + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + \mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Tag}}(\mathcal{C}).$$

To that point, the only way for the adversary to make π accept maliciously is to send a valid message τ'_B (resp. τ'_A) different from all the messages sent by all the instances. However, in such a case, the challenger aborts. Therefore

$$\Pr[E_4] = 0.$$

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

$$\begin{aligned}
\mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE}(\mathcal{A}) &= \Pr[E_0] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + \Pr[E_1] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq \times \Pr[E_2] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\left(\Pr[E_3] + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + \mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Tag}}(\mathcal{C})\right) \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\left(\Pr[E_4] + (q+1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + 2\mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Tag}}(\mathcal{C})\right) \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\left((q+1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + 2\mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Tag}}(\mathcal{C})\right) \\
&\leq nq\left((nq-1)2^{-\lambda} + (q+1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + 2\mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Tag}}(\mathcal{C})\right)
\end{aligned}$$

□

Key indistinguishability. Now we prove the key indistinguishability security.

Proof. Let E'_i be the event that an adversary win the key indistinguishability experiment in Game i, and $\mathsf{adv}_i = \Pr[E'_i] - \frac{1}{2}$.

Game 0. This game corresponds to the key indistinguishability experiment described in Chapter 2, Section 2.3.1. Therefore

$$\Pr[E'_0] = \frac{1}{2} + \mathsf{adv}^{\mathsf{key\text{-}ind}}_{SAKE}(\mathcal{A}) = \frac{1}{2} + \mathsf{adv}_0.$$

Game 1. In this game, we add an abort rule. The challenger aborts the experiment and chooses $b' \in \{0,1\}$ uniformly at random if there exists an instance 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

$$\mathsf{adv}_0 \leq \mathsf{adv}_1 + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE}(\mathcal{A}).$$

Game 2. In this game, we add an abort rule. The challenger tries to guess which instance is targeted by the adversary. If the guess is wrong, the game is aborted. The number of instances is at most nq. Therefore

$$\mathsf{adv}_2 = \mathsf{adv}_1 \times \frac{1}{nq}.$$

Game 3. Let π be the instance targeted by the adversary. We reduce the advantage of the adversary to win this game to the security of the function KDF used to compute the session key. That is, we rely upon the pseudo-randomness of the KDF function. This is possible if the key K is random. The genuine value of K is uniformly chosen at random by assumption. Then K is updated with update at most once per session on average. Therefore, when the u-th session starts, K has been updated at most $u-1$ times already. Therefore we must take into account the successive losses due to the key update with respect to the pseudo-randomness of update. Since there is at most q sessions per party (i.e., per original key K), this loss is at most $(q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B})$. Hence we replace each function $\mathsf{update}(K) = \mathsf{PRF}_{\mathsf{update}}(K,x)$ (keyed with a different key K throughout the, at most, $q-1$ successive sessions established, prior to that current session, by the same party that owns π) with truly random functions $\mathsf{G}^{\mathsf{update}}_0$, ..., $\mathsf{G}^{\mathsf{update}}_{q-2}$. Moreover, if an instance uses the same key $K = K_i, 0 \leq i < q-1$, to key update, then we replace update with the corresponding random function $\mathsf{G}^{\mathsf{update}}_i$. Since, to that point, the key $K = K_{q-1}$ used to compute the session key is random, we reduce the ability of $\mathcal{A}$ to win to the security of KDF. Therefore

$$\mathsf{adv}_2 \leq \mathsf{adv}_3 + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}(\mathcal{D}).$$

To that point the session key is random, therefore the adversary has no advantage in guessing whether $\pi.\mathsf{b} = b'$. That is

$$\mathsf{adv}_3 = 0.$$

Collecting all the probabilities from Game 0 to Game 3, we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{key\text{-}ind}}_{SAKE}(\mathcal{A}) &= \mathsf{adv}_0 \\
&\leq \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE}(\mathcal{A}) + \mathsf{adv}_1 \\
&\leq \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE}(\mathcal{A}) + nq \times \mathsf{adv}_2 \\
&\leq \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE}(\mathcal{A}) + nq\left(\mathsf{adv}_3 + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}(\mathcal{D})\right) \\
&\leq \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE}(\mathcal{A}) + nq\left((q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}(\mathcal{D})\right)
\end{aligned}$$

□

6.4 SAKE-AM: a Complementary Mode of SAKE

In SAKE the initiator A owns the three keys K'_{j+1}, K'_j, K'_{j-1}, and the responder B does the lightest computations. In this section, we present a variant of SAKE such that the smallest amount of calculations is done by the initiator. This variant corresponds also to less messages, and we call this "aggressive mode" of the protocol *SAKE-AM*.

Compared to SAKE, SAKE-AM inverts the role of the initiator and the responder in terms of calculations (in SAKE, the initiator performs – at most – two additional MAC computations compared to the responder). Thus B owns three samples of the authentication master key (corresponding to consecutive epochs), and A does the smallest amount of calculation. SAKE-AM (with one message less with respect to SAKE) allows computing the synchronisation gap δ earlier (with the first message). Yet the responder must wait for the third message to confirm that value. In a sense, this variant is also more optimistic.

The main idea is to skip the first SAKE message $A\|r_A$. Hence the roles between the two parties are swapped. This leads to other minor changes in message format compared to SAKE. Despite these differences, the messages and the calculations are essentially the same as in SAKE. This variant remains a sound and secure AKE protocol (according to Definition 2.10).

Furthermore, in a similar way to SAKE, the session in SAKE-AM can be reduced from four to three messages in some cases. Indeed, regarding the synchronisation state, in two cases (when $\delta_{AB} \in \{-1, 0\}$, that is $\epsilon = 0$), A and B are synchronised, and share a session key once A has received message m_B and executed the subsequent operations. Therefore, in such a case, the session can end upon reception of message τ'_A by B. More precisely

- if $\delta_{AB} = 1$ ($\epsilon = 1$), then A accepts upon reception of τ'_B, and B accepts upon reception of τ'_A;
- if $\delta_{AB} \in \{-1, 0\}$ ($\epsilon = 0$), then A accepts upon reception of m_B, and B accepts upon reception of τ'_A.

Notation. For the sake of clarity, we use the following notation in Figure 6.4:

- kdf corresponds to: $sk \leftarrow \mathsf{KDF}(K, f(r_A, r_B))$
- upd'_A corresponds to
  1. $K \leftarrow \mathsf{update}(K)$
  2. $K' \leftarrow \mathsf{update}(K')$
- upd'_B corresponds to
  1. $K \leftarrow \mathsf{update}(K)$
  2. $K'_{j-1} \leftarrow K'_j$
  3. $K'_j \leftarrow K'_{j+1}$
  4. $K'_{j+1} \leftarrow \mathsf{update}(K'_{j+1})$

End-device-server setting. Executing either SAKE or SAKE-AM depending on the party's role results in an implementation (gathering all the properties summarised in Section 6.2.1, starting with the forward secrecy property) that allows any party to be either initiator or responder of a session, and such that the smallest amount of calculation is always done by the

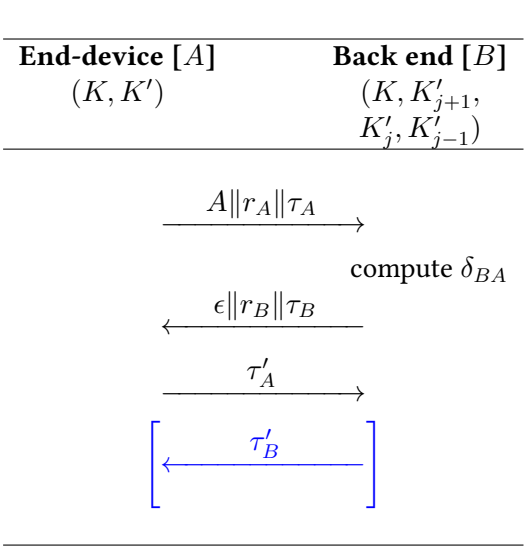
A (K, K')		B $(K, K'_{j+1}, K'_j, K'_{j-1})$
$r_A \xleftarrow{\$} \{0,1\}^\lambda$ $\tau_A \leftarrow \mathsf{MAC}(K', A\|B\|r_A)$ $m_A \leftarrow A\|r_A\|\tau_A$	$\xrightarrow{m_A}$	
		`if` $(\mathsf{Vrf}(K'_j, A\|B\|r_A, \tau_A) = \mathtt{true})$ $\delta_{BA} \leftarrow 0$ $K' \leftarrow K'_j$; kdf; upd'_B; $\epsilon \leftarrow 0$ `else if` $(\mathsf{Vrf}(K'_{j-1}, A\|B\|r_A, \tau_A) = \mathtt{true})$ $\delta_{BA} \leftarrow 1$ $K' \leftarrow K'_{j-1}$; $\epsilon \leftarrow 1$ `else if` $(\mathsf{Vrf}(K'_{j+1}, A\|B\|r_A, \tau_A) = \mathtt{true})$ $\delta_{BA} \leftarrow -1$ $K' \leftarrow K'_{j+1}$; upd'_B; kdf; upd'_B; $\epsilon \leftarrow 0$ `else` abort
		$r_B \xleftarrow{\$} \{0,1\}^\lambda$ $\tau_B \leftarrow \mathsf{MAC}(K', \epsilon\|B\|A\|r_B\|r_A)$ $m_B \leftarrow \epsilon\|r_B\|\tau_B$
	$\xleftarrow{m_B}$	
`if` $(\mathsf{Vrf}(K', \epsilon\|B\|A\|r_B\|r_A, \tau_B) = \mathtt{false})$ abort		
`if` $(\epsilon = 1)$ upd'_A kdf; upd'_A $\tau'_A \leftarrow \mathsf{MAC}(K', A\|B\|r_A\|r_B)$		
	$\xrightarrow{\tau'_A}$	
		`if` $(\epsilon = 0)$ $K' \leftarrow K'_j$ `if` $(\mathsf{Vrf}(K', A\|B\|r_A\|r_B, \tau'_A) = \mathtt{false})$ abort `else if` $(\epsilon = 1)$ $K' \leftarrow K'_{j+1}$ `if` $(\mathsf{Vrf}(K', A\|B\|r_A\|r_B, \tau'_A) = \mathtt{false})$ abort kdf; upd'_B $\tau'_B \leftarrow \mathsf{MAC}(K', r_B\|r_A)$
	$\xleftarrow{\tau'_B}$	
`if` $(\mathsf{Vrf}(K', r_B\|r_A, \tau'_B) = \mathtt{false})$ abort		

Figure 6.4 – SAKE-AM protocol

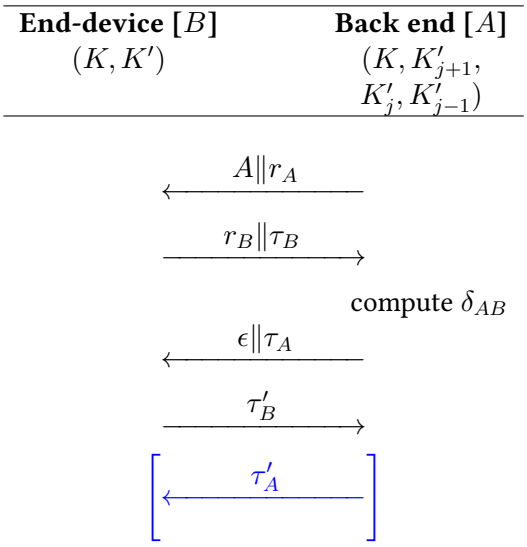
same party. Furthermore, SAKE and SAKE-AM are built on the same cryptographic functions, which enables to pool their implementation. This is particularly convenient in the context of a set of (low-resource) end-devices communicating with a central server. In such a case, the end-device supports the lightest computations, whereas either the server or the end-device can initiate a session.

Figure 6.5 illustrates such a configuration. When the end-device initiates a session, protocol SAKE-AM is applied. Otherwise (the server is initiator), SAKE is executed.

Remark. In Chapter 7 we introduce a 3-party key exchange protocol dedicated to IoT, and describe how SAKE and SAKE-AM can be appropriately used in such a context.



(a) End-device is initiator (SAKE-AM)



(b) Back end is initiator (SAKE)

Figure 6.5 – SAKE/SAKE-AM executed between a low-resource end-device and a back-end server. Both parties may initiate the session. In some cases, the last message can be skipped.

Soundness and security of SAKE-AM. SAKE-AM is a sound and secure AKE protocol. The proofs for SAKE-AM follow the same reasoning and are similar to that of SAKE. They yield the same bounds (see Section 6.3).

6.5 A Random-free Variant of SAKE

In SAKE (and SAKE-AM), the pseudo-random values r_A, r_B are used to yield a fresh session key, and participate also in the authentication of the parties. Using new values during each session contributes to achieving these two tasks. Yet, these parameters are not the only ones to evolve throughout the successive protocol runs. The master keys do also. Therefore, one can consider removing the pseudo-random values from the messages. Without the pseudo-random values, several messages become cryptographically valid for each flow (instead of one only in SAKE). For instance, without r_A, party A may accept as second message either $\tau_B = \mathsf{MAC}(K'_j, B\|A)$,

or $\tau_B = \mathsf{MAC}(K'_{j-1}, B\|A)$, or $\tau_B = \mathsf{MAC}(K'_{j+1}, B\|A)$. Likewise, without r_B, B may accept as third message either $0\|\tau_A$ or $1\|\tau_A$. Consequently, in this variant, we prefix each MAC-ed message with its index from 1 to 4 (but not the first one which carries only the initiator's identity).

The removal of the pseudo-random values enables a "mismatch attack". By "attack" we mean the following: an adversary is able to compel B to compute a message (message 4) which is *unaltered* by the adversary and expected by A, and yet A rejects this message as invalid. Although unpleasant, this "attack" does not break any claimed security property (in particular entity authentication). Moreover, this scenario cannot damage the synchronisation of the two parties. That is, if they start a new session, the latter completes successfully (if the adversary remains passive), as in SAKE.

In this variant, the length of the messages is shortened, and this avoids also calling the pseudo-random generation function. This is advantageous for low-resource devices. Therefore, according to us this variant is suitable for constrained devices. Nonetheless, not using pseudo-random values, in particular to derive the session key, may not be of anyone's taste. In addition, the possibility provided by the aforementioned scenario is not what one usually expects from a security protocol. Consequently, for the practitioners for whom this mismatch attack is unacceptable, the SAKE protocol is more adequate.

6.6 Comparison with the DH Paradigm

The protocol SAKE (as well as SAKE-AM) is based on shared master keys and apply symmetric-key functions only. In particular it does not require the application of any kind of DH-like scheme. Yet it provides a strong form of forward secrecy. Despite this result, our protocol differs from a DH scheme in several ways beyond the intrinsic distinction between public-key and symmetric-key cryptography.

Concurrent executions. Our protocol does not allow parallel executions. Indeed, since it is based on shared evolving symmetric keys, running multiple instances in parallel may cause some sessions to abort. A way to relax this restriction is that each party use separate master keys for concurrent executions. On the contrary, the DH scheme allows an (virtually) unlimited number of parallel executions.

KCI attacks. The ephemeral DH scheme (when using safe parameters) is resistant against KCI attacks [BWJM97], whereas our protocol is not (due to the dependency between the (updated) master keys).[5] Moreover if an adversary succeeds in getting the key K' (or K'_j), she can compute the subsequent key (corresponding to K'_{j+1}). Hence the adversary can forge a message m_B in SAKE that brings the initiator to update its master keys twice consecutively. Therefore, that party is desynchronised with respect to an honest partner, with no possibility to resynchronise.

Note that KCI attacks affect also the static DH scheme (when a party uses a fixed DH share, whereas the other generates a fresh ephemeral one [HGFS15]).

Another consequence of the dependency of the master keys in SAKE, is that once the keys are revealed, an adversary can passively compromise all subsequent session keys. This is not the case in general with ephemeral DH. Yet, this is also true regarding non-DH public-key protocols (e.g., TLS-RSA), but also ephemeral DH (in some pathological cases) when small, fixed public parameters are used [ABD+15].

[5] When a party P_i's long-term secret key is disclosed, an adversary can impersonate P_i to other parties. In the same context, a key compromise impersonation (KCI) allows the adversary to impersonate other parties *to* P_i.

Post-quantum setting. Now a probable benefit of our protocol compared to the DH scheme is that, since it is based on symmetric-key functions, it can likely survive in a post-quantum world (with a suitable choice of the primitives [KLLN16a] and key length [Gro96]). On the contrary, the DH scheme is known to be insecure in such a context [Sho94; PZ03; KJ17]. Yet, we observe that there exists a post-quantum variant of the original DH scheme [JD11; CLN16], but it is based on larger parameters and heavier computations than SAKE. Moreover this post-quantum variant does not provide entity authentication.

Computations. The DH scheme implies heavier computations (modular exponentiations, elliptic curve point multiplication) than SAKE which is solely built on symmetric-key functions. In practice, SAKE is likely more suitable to be implemented on constrained devices which have limited computational (and communication) capabilities.

7 Three-party AKE

With the rise of the Internet of Things several security protocols are widely deployed or strongly promoted, such as LoRaWAN, Sigfox, and NB-IoT among others. They aim at connecting with one another multiple components which have different capabilities (low-resource end-devices, powerful servers).

In Chapters 3 and 4, we have analysed protocols dedicated to constrained devices, and show that they suffer from several weaknesses leading to (likely) practical attacks. In Chapter 6, we have presented the 2-party authenticated key exchange protocol SAKE with stronger security properties. In this chapter, we widen our vision and consider the interleaved operations between the diverse components of an IoT network. Consequently, we describe a generic 3-party authenticated key exchange protocol dedicated to such a network. Solely based on symmetric-key functions (regarding the computations done between the end-device and the back-end network), this protocol guarantees forward secrecy, in contrast to widely deployed symmetric-key based IoT protocols. Furthermore, it enables session resumption without impairing security (in particular, forward secrecy is maintained). This allows saving communication and computation cost, and is advantageous for constrained end-devices.

In addition, we present a concrete instantiation of our protocol based on SAKE, and devise a security model used to formally prove the security of the protocol and its instantiation.

The 3-party key exchange protocol can be applied in a real-case IoT deployment (i.e., involving numerous end-devices and servers) such that the latter inherits from the security properties of the protocol. This results in the ability for a (mobile) end-device to securely switch from one server to another back and forth at a reduced (communication and computation) cost, without compromising the sessions established with other servers.

The results of this chapter have been published in [ACF19].

Contents

7.1 Introduction

7.1.1 Context

The arising of the Internet of Things (IoT) gives birth to different types of use cases and environments (smart home, smart cities, eHealth, Industrial IoT, etc.). According to several reports, "*the Industrial Internet of Things is the biggest and most important part of the Internet of Things*" [i-s18] and "*the biggest driver of productivity and growth in the next decade*" [Acc15]. The Industrial IoT (IIoT) covers sensitive applications since it aims at managing networks that provide valuable resources (e.g., energy, water, etc.). Contrary to the smart home case, where a network is localised to the house perimeter and implies merely a domestic management of the network, the IIoT context may require a large coverage zone where connected objects (e.g., sensors, actuators, etc.) are widespread all over an urban area. This implies the involvement of, at least, two players: the application provider (which exploits the connected objects to get some valuable data and provide some service), and the communication provider whose network is used by the application provider to communicate with its connected objects (see Figure 7.1). These two entities needs to communicate with each end-device for different purposes, which implies two independent secure channels.

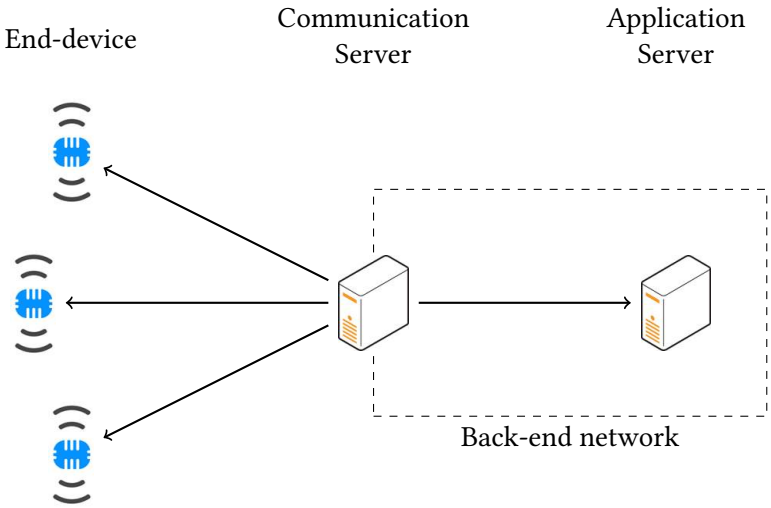


Figure 7.1 – Connection between end-devices (ED ←) and an application server (→ AS) through a communication server (– CS –)

Furthermore, as indicated in Chapter 1, the protocols for the (Industrial) IoT build their security on symmetric-key functions, and make use of a static and unique (per end-device) root key shared between the end-device and the back-end network. To this class of protocols belong widely deployed or strongly promoted ones such as Sigfox [Sig17b; Sig17a], LoRaWAN [LoR18a; Sor17], and cellular technologies such as Narrowband IoT (NB-IoT), enhanced Machine-Type Communication (eMTC), Extended Coverage GSM IoT (EC-GSM-IoT). None of these (cellular and non-cellular) protocols provide forward secrecy, and, to the best of our knowledge, no IoT protocol proposes a session resumption scheme.

Cryptographic separation of the layers. The (Industrial) IoT involves low-resource end-devices which are not able to apply heavy computations implied by asymmetric schemes. Consequently, security protocols used on currently deployed IoT networks usually implement

symmetric-key functions only, and are based on a unique (per end-device) symmetric root key. Using the same root key implies that the communication layer and the application layer are entangled. The communication provider must guarantee that only legitimate parties can send data through its network, but does not need to get the application data. The application provider must keep full control over its connected objects, but must not be able to interfere with the management of the communication network. Therefore, the communication and the application layers must be cryptographically distinct.

Forward secrecy. The (Industrial) IoT protocols based on symmetric-key functions do not provide strong security properties usually ensured by asymmetric schemes, in particular *forward secrecy*. The disclosure of the root key compromises all the past sessions established with that key, not to mention the consequences of an intrusion into the back-end server that centralises all root keys. The current symmetric-key based IoT protocols lack in providing this fundamental security property.

Session resumption. A session resumption scheme allows establishing a new session at a reduced cost: once two parties has performed a first key exchange, they can use some shared key material to execute subsequent runs faster. This means less data exchanged during the key agreement, and reduced time and energy, which is particularly convenient and advantageous for low-resource end-devices. Yet, the symmetric-key based IoT protocols always execute the same full key exchange.

7.1.2 Our Approach

The basis of our approach is the need for an authenticated key exchange protocol guaranteeing better security properties than that of existing IoT protocols (including widely deployed ones), and being at the same time suitable for low-resource end-devices. Consequently, we consider the symmetric-key setting. In order to cryptographically separate the communication and the application layers, we consider a LoRaWAN-like architecture with a trusted third party behaving as a key server (see Chapter 3, Section 3.5). But we use the latter in a more efficient way. We describe an authenticated key exchange protocol that involves three parties: an end-device, a (communication or application) server, and a trusted third party (the key server). This protocol is solely based on symmetric-key functions (regarding the computations done by the end-device), and yet it provides forward secrecy.

Moreover our 3-party protocol allows resuming sessions. A full run implies the involvement of the trusted third party in order to establish a session between an end-device and a (communication or application) server. The resumption procedure allows executing the subsequent runs between the end-device and the same server without the trusted third party being involved. That is, all the messages exchanged with the latter are saved. This means a faster session establishment with reduced time and energy (in particular for the low-resource end-device). Our resumption scheme implies a small and fixed size storage in the end-device's memory independently of the number of sessions that can be resumed. Finally, our protocol allows resuming sessions without impairing security: it combines session resumption and forward secrecy.

An IoT network involves many entities and not only a pair of end-device and server. Our 3-party key exchange protocol can be used to deploy an arbitrary number of end-devices and (communication or application) servers within the same network (i.e., affiliated to the same trusted third party). This allows in particular a (mobile) end-device to switch from one com-

munication server to another one, or to be used by two different application servers without compromising sessions established with other servers.

To sum up, in this chapter we present a 3-party authenticated key exchange (3-AKE) protocol executed between an end-device, a server, and a trusted third party, which matches *at the same time* the following properties:

- The protocol is solely based on symmetric-key functions (regarding the computations done by the end-device).
- Application and communication security layers are separated.
- The protocol enables session resumption.
- The protocol provides forward secrecy.

7.2 Description of the 3-party AKE Protocol

In this section, we describe our generic 3-party authenticated key exchange (3-AKE) protocol. The main purpose of our protocol is to output session keys. This subsequently enables to establish two distinct secure channels, with a communication server on the one hand, and an application server on the other hand. We do not detail these channels, and let it be defined depending on their specific context.

7.2.1 The Different Roles

The real-case IoT deployment we consider involves four *roles*: the trusted third party that we name *Authentication and Key Server* (KS), the *Application End-device* (ED), the *Communication Server* (CS), and the *Application Server* (AS). KS is in charge of the overall security of the system: its main purpose is to authenticate ED, and to allow AS and CS to share distinct session keys with ED. These keys aim at establishing two separate secure channels. The purpose of AS is to provide some service (e.g., telemetry, asset tracking, equipment automation, etc.). The AS exploits ED (e.g., a sensor, an actuator, etc.) to ensure that service. In order to exchange confidential data, ED and AS use a communication network. The entry point is CS, which grants ED access to that network. Typically CS is managed by a telecom operator. CS needs also to privately communicate with ED, e.g., in order to regulate the radio interface.

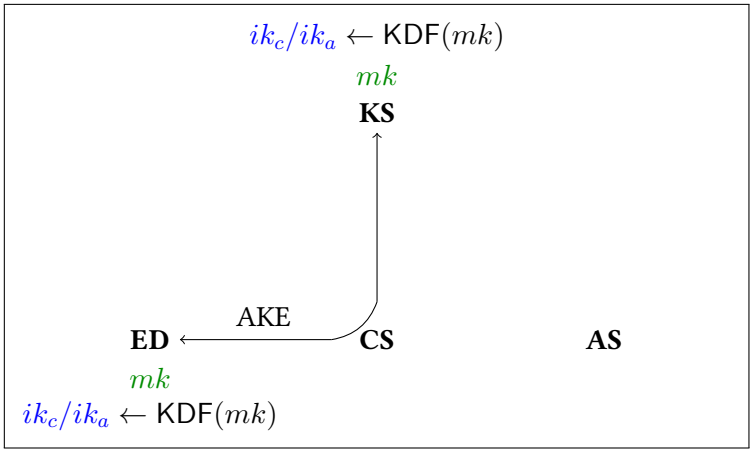
One KS can manage several ED. An ED can be either static or mobile, hence may have to connect one or several CS. An AS can use several ED in order to provide its service. For the sake of genericness, it may also be technically (i.e., cryptographically) possible with our protocol that the same ED be used by several AS (each one providing a different service). The kind of ED we consider is a (low-resource) wireless end-device whereas we assume that KS, CS, and AS use faster (wired) connections with each other, and have heavier capabilities, in particular computational.

As said, the architecture we consider involves four types of entities: KS, ED, CS, and AS. However, from a cryptographic perspective, CS and AS behave the same way with respect to KS and ED. The main goal to reach is to allow ED to share a session key with a server XS $\in \{\text{CS}, \text{AS}\}$ which ensures some functionality (communication or application in our case). This is achieved with our 3-AKE protocol: executed between KS, ED, and XS, the protocol outputs key material that allows ED and XS to establish a secure channel. In the remainder of the chapter, we will mention for simplicity only the two types of CS and AS servers. Nonetheless, recall that they

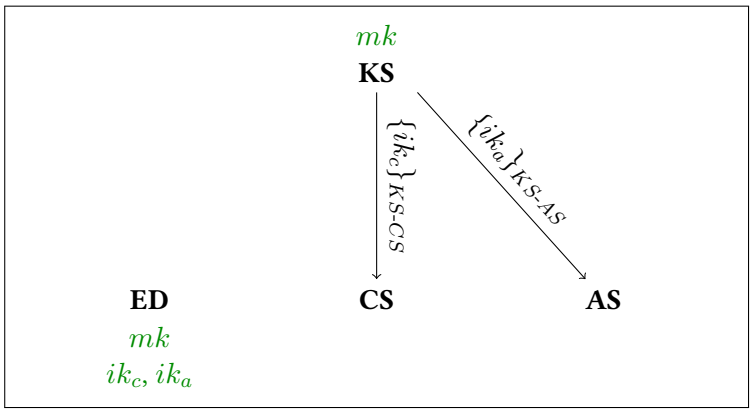
represent in fact the several servers which are actually involved in the IoT architecture we consider.

7.2.2 Key Computation and Distribution

Our 3-AKE protocol is based on a pre-shared symmetric key mk known only to two parties: ED, and KS which ED is affiliated to. Each ED owns a distinct master key mk. A 3-AKE run is split in two main phases. Each phase appeals to a 2-party authenticated key exchange (2-AKE) protocol, whose security properties will be made explicit in Section 7.2.3. During the first phase (Figure 7.2), ED and KS perform a 2-AKE run with the shared master key mk. During the second phase (Figure 7.3), ED and XS $\in \{\text{CS}, \text{AS}\}$ use the output of the first key exchange to perform an additional 2-AKE run. This yields a session key used to establish a secure channel between ED and XS. In practice, since our architecture involves two types of XS servers, a 3-AKE run is done first between KS, ED, and CS, and then between KS, ED, and AS. This yields two distinct session keys. With each session key, a secure channel can be established between ED and CS on the one hand, and ED and AS on the other hand.



(a) 2-AKE run executed between ED and KS (relayed by CS) with mk



(b) Transmission by KS of intermediary keys ik_c (to CS) and ik_a (to AS) respectively through the secure channels $\{\cdot\}_{KS\text{-}CS}$ established between KS and CS, and $\{\cdot\}_{KS\text{-}AS}$ established between KS and AS

Figure 7.2 – 2-AKE run executed between ED and KS with mk, and distribution of ik_c, ik_a

More precisely, the following steps are executed between KS, ED, CS, and AS.

1. Based on the shared master key mk, KS and ED perform an AKE, relayed by CS (Figure 7.2a). This first AKE outputs a *communication intermediary key* ik_c.
2. The previous step (2-AKE) is repeated between KS and ED. It outputs an *application intermediary key* ik_a.
3. KS sends ik_c to CS, and ik_a to AS through two distinct pre-existing secure channels (Figure 7.2b). Then, upon reception of the keys by CS and AS, KS deletes its own copies in order to enhance the security of the subsequent phases of the protocol (we elaborate more on this in Section 7.2.4).
4. Using ik_c, ED and CS perform an AKE which outputs a *communication session key* sk_c (Figure 7.3a).
5. Using ik_a, ED and AS perform an AKE which outputs an *application session key* sk_a (Figure 7.3b).
6. Using the application session key sk_a, ED and AS can now establish an *application secure channel*. Likewise, with the communication session key sk_c, ED and CS can establish a distinct *communication secure channel* (Figure 7.3c).

We call P the protocol that involves ED, and is used to perform the 2-AKE runs between ED and KS (steps 1-2), ED and CS (step 4), and ED and AS (step 5). We call P' the 2-AKE protocol used on the back-end side between KS and AS (resp. CS). Let ENC be the function used to set up the secure channel between KS and AS (resp. CS) with the session key output by P'. That is, ENC is used (step 3) to encrypt ik_a (resp. ik_c) prior to being sent by KS to AS (resp. CS).

Variants. For the sake of clarity, we have depicted in Figure 7.2a the case where the two intermediary keys ik_c and ik_a are successively computed. But the computation of either key can be completely dissociated. Conversely, it may also be possible that both keys be computed at once during the same run. The same key exchange protocol can be used in either case, the difference lying in an additional derivation step that yields two keys from the unique output of the original 2-party AKE.

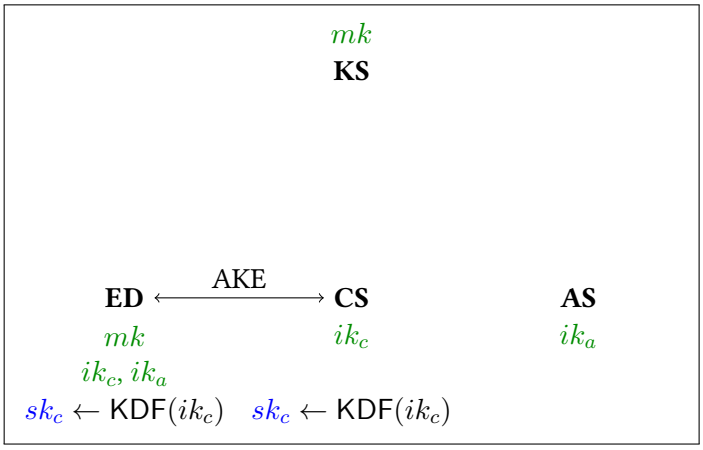
Furthermore, the 2-AKE run between ED and KS can output not only an intermediary key ik but also a session key sk. Then KS sends both ik and sk to XS. This saves an AKE run between ED and XS (corresponding to steps 4-5).

7.2.3 The Building Blocks P, P', and ENC

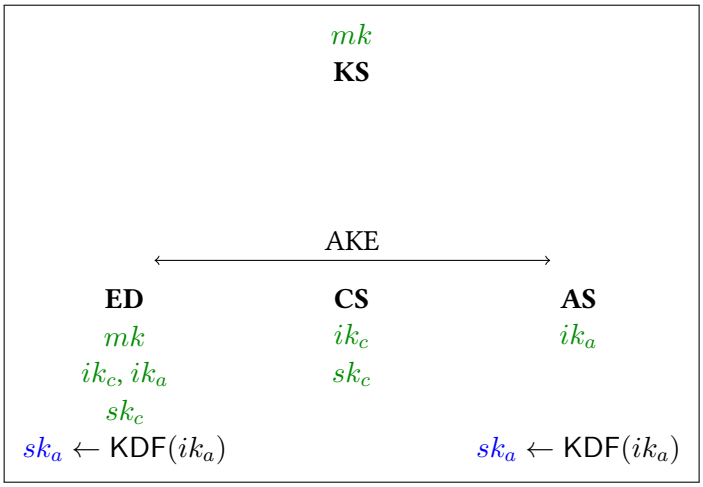
Our 3-AKE protocol depends crucially on the 2-party protocols P and P', and function ENC. Before making clear the properties of our 3-party protocol, we list below the main features we require these three building blocks to have.

Protocol P. We require protocol P to fulfill the following properties.

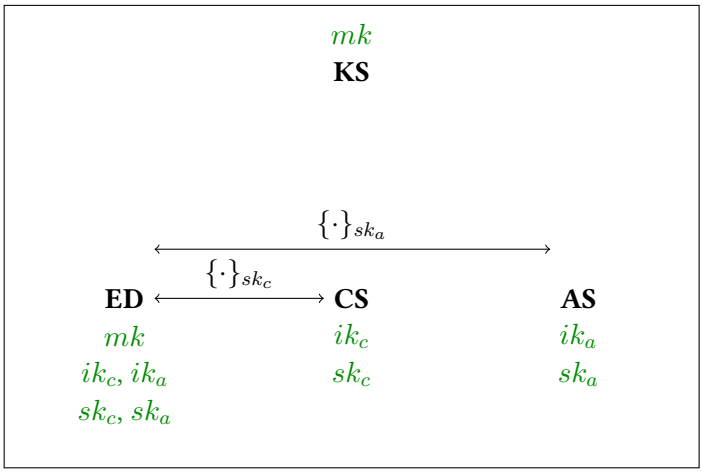
- The scheme is a 2-party AKE protocol that provides mutual authentication.
- The scheme is based on symmetric-key functions solely.
- The scheme guarantees forward secrecy.



(a) 2-AKE run executed between ED and CS with ik_c



(b) 2-AKE run executed between ED and AS (relayed by CS) with ik_a



(c) Secure channels established: communication channel $\{\cdot\}_{sk_c}$ between ED and CS, and application channel $\{\cdot\}_{sk_a}$ between ED and AS

Figure 7.3 – 2-AKE run executed between ED and AS (resp. CS) with ik_a (resp. ik_c), and subsequent secure channels

Although it is not related to the main goals we tackle, we add the following requirement in order to improve the flexibility of the 3-AKE protocol:

- Any of the two parties can initiate a run of protocol P.

We informally recall what the forward secrecy property means in this symmetric-key context. Once a 2-AKE run of P is complete, past output secrets must remain private even if the current symmetric root key (used to authenticate the parties and compute the shared secret) is revealed.

More precisely, in a 2-AKE run done between ED and KS, the disclosure of the current master key mk (used as root key) must not compromise past intermediary keys ik computed by these two parties. Likewise, in a 2-AKE run done between ED and some XS $\in \{\text{CS}, \text{AS}\}$, the disclosure of the current intermediary key ik (used as root key in that case) must not compromise past session keys sk computed by ED and XS.

In Section 7.4 we present a concrete instantiation of P based on the SAKE protocol that we have introduced in Chapter 6.

Protocol P'. We demand P' to be a secure 2-AKE protocol that provides mutual authentication, and forward secrecy. Since P' is applied between KS and XS, asymmetric functions may be used.

Function ENC. We demand ENC to provide data confidentiality and data authenticity. In the latter we include non-replayability of messages.

7.2.4 Main Features of the 3-AKE Protocol

In Chapter 5, Section 5.2, we have presented a 3-AKE security model that formally defines the properties we demand a 3-AKE protocol to ensure. In Section 7.5.1 we use this security model to prove that the three main components P, P' and ENC yield a secure 3-AKE protocol. Before, we detail in this section the main features provided by our 3-AKE protocol and informally justify these properties.

Management of the security. The key hierarchy (between mk, ik, and sk), allows ED and KS to manage the overall security of the system. The key exchange done between KS and ED (steps 1-2, Section 7.2.2) can be initiated by any but only these two entities. Each 2-AKE done between ED and KS creates a new intermediary key ik. This obsoletes the current intermediary key shared by ED and XS $\in \{\text{CS}, \text{AS}\}$, and "disconnects" ED from XS by resetting ik at ED. Hence, KS and ED can defend against a dishonest or corrupted XS.

Cryptographic separation of the layers. The use of two distinct intermediary keys ik_c and ik_a allows separating the communication layer (between ED and CS) and the application layer (between ED and AS). The mutual authentication done between KS and, respectively, CS and AS, guarantees that the intermediary keys are sent to and received from legitimate parties only.

Secure connection to any server. The 3-AKE protocol allows ED to share an intermediary key ik with any (communication or application) server. Moreover, ED can connect any such server without impairing the security with another server. First, each 2-AKE run done between ED and KS yields a different intermediary key ik. Hence each partnered ED and XS use a distinct key ik. Next, KS deletes its copy of ik as soon as it has been received by XS. Finally, P provides

forward secrecy. The disclosure of the current master key mk (stored at ED and KS) does not compromise a past output key ik. The forward secrecy ensured by P' and the security of the channel established with ENC participate also in the confidentiality of ik. Likewise, due to the forward secrecy of P, past session keys sk (computed between ED and XS) remain private, even if the current key ik (stored at ED and XS) is exposed

Quick session establishment. Once a first intermediary key ik is shared between ED and XS, these two parties can perform as many 2-AKE runs (hence set up as many successive secure channels) as wished *without* soliciting KS anymore (i.e., ED and XS repeat several times step 4 or 5, Section 7.2.2). This avoids overloading KS (which has to manage many ED and XS). An additional consequence is also that the number and the frequency of the connections established between ED and XS are hidden from KS.

7.3 Session Resumption Procedure

In this section, we explain how to shorten the genuine key exchange procedure of the 3-AKE protocol such that ED and CS or AS, after a first successful protocol run involving KS, can execute subsequent key exchanges without the need for KS to intervene.

7.3.1 Rationale for a Session Resumption Procedure

As explained in Section 7.2.4, after a first 2-AKE run with KS, ED shares an intermediary key ik with XS $\in \{\text{CS}, \text{AS}\}$. Then, ED and XS can execute, from ik, subsequent 2-AKE runs without soliciting KS anymore. Consequently, as soon as ED shares (distinct) intermediary keys with several servers, it can quickly switch from one server to another back and forth without the help of KS. This is particularly convenient for a mobile ED which must connect different communication providers (hence different CS servers). Likewise, this allows ED to connect several AS servers, hence to be securely used by different application providers. Moreover, since P guarantees forward secrecy, the disclosure of (the current value of) ik does not compromise past session keys sk. We call this faster mode (without KS) a *session resumption* procedure.

Due to the intrinsic properties of the 2-AKE scheme P (see Section 7.2.3), any peer (ED or XS) can initiate the key exchange. This implies that *both* peers can initiate the session resumption procedure.

The main benefit of this procedure is to give the ability to switch between servers without soliciting KS. Avoiding the involvement of KS (that is, avoiding a whole 2-AKE run between ED and KS), allows to save time, computation cost and communication cost for KS but mainly for ED. Indeed, the ED we consider are low-resource, self-powered devices. The energy cost to transmit and to receive data usually exceeds the cost of cryptographic processing [SP05; WGE+05]. Hence it is worth saving as much as possible the amount of data exchanged to compute a new session key.

Another limitation of a low-resource ED is its memory space. Being able to resume a session with several servers implies storing simultaneously as many intermediary keys. This is likely possible for a server but becomes prohibitive for such kind of ED. In Section 7.3.2, we present a session resumption scheme that solves this issue.

7.3.2 Session Resumption Procedure for Low-resource ED

Overview of the procedure. The session resumption procedure for a low-resource ED with XS $\in \{\text{CS}, \text{AS}\}$ is made of two phases:

(a) The *storage phase*. ED and XS have an ongoing secure channel set up with a session key sk (output by P). Both share an intermediary key ik. First, ED encrypts ik under a key known only to itself (we elaborate on this below). Next, ED sends the resulting "ticket" to XS through the ongoing secure channel. Upon reception of the ticket by XS, ED deletes ik. Then ED can close the channel any time.

(b) The *retrieval phase*. ED starts a new 2-AKE run with a known XS. First, ED gets, in the continuity of the run, the ticket it has sent previously. Next, ED decrypts the ticket and gets the corresponding key ik. Then, ED and XS complete the run with ik, and compute a new session key sk.

This procedure is reminiscent of existing schemes (e.g., [SZET08; Res18; ST10]), yet intended to a different context than IoT. However none of the latter succeeds in combining session resumption and forward secrecy without asymmetric cryptography or prohibitive requirements (for a constrained ED) regarding memory, or the amount of transmitted data [Res18; AGJ19]. In Section 7.3.3 we compare our proposal with other resumption schemes. In contrast, our 3-AKE protocol provides a shrewd solution to this issue, as explained below.

Computing the ticket. The intermediary key ik that is stored at the server and later retrieved by ED is encrypted. Only ED needs to decrypt ik since the server stores its own copy of the key. Using the same encryption key k to protect different intermediary keys (sent to different servers) obviously breaks forward secrecy: revealing k allows decrypting *past* intermediary keys, hence compromising the session keys sk computed with the latter. Therefore each intermediary key must be encrypted with a different key k. However, replacing in ED's memory each intermediary key ik with another (encryption) key k yields the same memory issue and is pointless. Therefore, we compute the keys k used to encrypt the intermediary keys as elements of a *one-way key chain*.

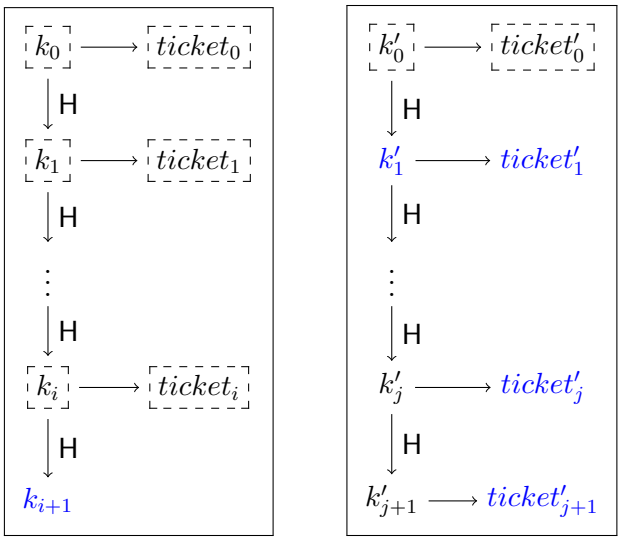
From an initial random key k_0, each ticket is computed as $ticket_{i+1} = \mathsf{KW}(k_{i+1}, ik)$ with $k_{i+1} = \mathsf{H}(k_i)$, $i \geq 0$. KW is a key-wrap function [RS06], and H a one-way function. ED keeps in memory only one key k_j. This key is the child of the key that has decrypted the last used ticket. When ED wants to consume $ticket_i$, it first computes the decryption key k_i from the current key k_j, $i \geq j$: $k_i = \mathsf{H}^{i-j}(k_j)$. Then k_j is replaced with $k_{i+1} = \mathsf{H}(k_i)$, and $ticket_i$ *cannot* be decrypted anymore.

This *unique* encryption key gives ED the ability to compute multiple tickets, therefore to resume as many sessions.

Two chains of keys. When $ticket_i$ is used, the current decryption key is replaced with $k_{i+1} = \mathsf{H}(k_i)$. Hence any previous $ticket_j$, $j \leq i$, is obsoleted. Let us consider the following scenario. A mobile low-resource ED is managed by one AS, and switches back and forth between two other servers CS_a and CS_b. ED stores fresh $ticket_i$, $ticket_j$, and $ticket_k$, $i < j < k$, respectively at AS, CS_a, and CS_b. ED keeps the decryption key k_i. When ED makes a new key exchange with CS_a, it retrieves $ticket_j$ and decrypts it with $k_j = \mathsf{H}^{j-i}(k_i)$. Then, ED replaces the current key k_i with $k_{j+1} = \mathsf{H}(k_j)$. Whenever ED alternates between CS_a and CS_b, the ticket decryption key is updated. Consequently, ED cannot compute again k_i and decrypt $ticket_i$ which becomes unusable. Even though $ticket_i$ was the most recent ticket, it would be obsoleted at some point. This makes the session resumption procedure unusable with AS. Therefore, we advocate the use of two chains of decryption keys corresponding to the *two types* of CS and AS servers, and the two possibly different behaviours of ED (see Figure 7.4). Nonetheless, if a different context requires so, a unique chain of decryption keys can also be maintained.

If the tickets are used in the same order they are computed, all can be (legitimately) decrypted. In particular, according to us, it is likely, that ED, when alternating between several CS servers (or, equivalently, between several AS servers) uses the corresponding tickets accordingly.

Figure 7.4a depicts the case where a CS ticket ($ticket_i$) is used. The corresponding decryption key k_i is deleted, and ED keeps only k_{i+1}. This obsoletes all previous CS tickets. Figure 7.4b depicts the case where an AS ticket ($ticket'_0$) is used. The decryption key k'_0 is deleted, and ED keeps only k'_1. All AS $ticket'_j$, $j \geq 1$, are still usable.



(a) CS tickets and keys **(b)** AS tickets and keys

Figure 7.4 – Chains of keys used to compute a ticket. The tickets already used cannot be decrypted because the corresponding key cannot be computed anymore. The used tickets and their key appear in a dashed box. The current key and the available tickets appear in blue.

Maintaining forward secrecy. When $ticket_i$ is used, the current encryption key k_j, $j \leq i$, stored at ED, is replaced with the next encryption key $k_{i+1} = \mathsf{H}(k_i)$. This forbids any old ticket from being decrypted. All the remaining tickets that can be decrypted (from the now current key k_{i+1}) have not been used yet. Moreover, the protocol P provides forward secrecy. Hence, the disclosure of the intermediary key ik protected into a (not used yet) ticket does *not* compromise past session keys sk. In a way, the session resumption procedure inherits the forward secrecy from P (and also the one-wayness of function H). This property is formally proved in Section 7.5.2.

The use of the (forward secret) intermediary key ik highlights also why encrypting the session key sk in the ticket is not a good choice. The more data the same session key protects, the worse its disclosure.

Remark. Another reason to opt for ik is efficiency. In fact, sk may be quite large (e.g., two pairs of keys, encryption and MAC, for each direction, and the last value of the uplink and downlink frame counters). The ticket is transmitted twice between ED and XS. As explained above, the amount of data exchanged with the server is a burden for a wireless low-resource ED. From a single intermediary key ik, any kind of security parameters can be computed. Hence the choice

of ik.

7.3.3 Comparison with Other Session Resumption Schemes

To the best of our knowledge, no IoT protocol proposes a session resumption scheme. Nonetheless, such schemes exist in other contexts.

In TLS 1.2 [DR08], the server can encrypt the "master secret" and store that "Session Ticket" [SZET08] at the client. In TLS 1.3 [Res18], the server encrypts a "resumption master secret" (RMS) output by the previous key exchange, and stores it at the client. In IKEv2 [KHN+14], a similar approach is used [ST10].

From the same secret value, used as symmetric master key, successive runs can be executed with these (TLS, IKE) procedures. Hence disclosure of the reused secret may compromise several past sessions: this breaks forward secrecy. In TLS 1.3, a fresh secret can be added to the key derivation computation, but this implies applying the Diffie-Hellman scheme [DH76]. Moreover, in TLS, the same Session Ticket Encryption Key (STEK) is used by the server to encrypt several RMS values (corresponding to different clients). Hence a STEK may be persistent in the server's memory and its disclosure compromises past sessions. Therefore these solutions are not satisfactory with respect to forward secrecy.

Aviram, Gellert, and Jager [AGJ19] propose a resumption scheme aiming at guaranteeing forward secrecy and non-replayability when 0-RTT is used in TLS 1.3. They describe two concrete instantiations. One is based on RSA [RSA78], the other is a tree-based scheme. As all the aforementioned session resumption schemes, Aviram et al.'s proposal implies storing a ticket at the client. Therefore the number of tickets to store grows with the number of servers the client can resume a session with. Hence, low-resource end-devices with constrained memory cannot apply these schemes.

Reversing the roles taken by the client and the server (i.e., the client computes and the server stores the ticket) is not sufficient. First, the Aviram et al.'s RSA based scheme is excluded, despite its elegance, for low-cost IoT end-devices that can only implement symmetric-key functions. Moreover, their tree-based scheme implies that the decryption key grows (up to some point) each time a ticket is used, which is prohibitive for the end-device (client).

The two schemes (RSA- and tree-based) described by Aviram et al. allow computing a fixed number of tickets (say n). One key (asymmetric or symmetric depending on the scheme) is used to yield the n tickets. In order to compute a new batch of n tickets, a new key must be generated and stored by the server. We observe that, if a new batch of n tickets is computed whereas it remains even one ticket not used yet from the previous batch, two keys (the current and the new one) must be stored concurrently in the server's memory.

Aviram et al. propose also an alternative that trades decryption key size for ticket size. However sending (and retrieving) big tickets is an issue for a low-resource end-device. As noticed by Aviram et al., each transmitted bit costs energy, which limits the battery lifetime of self-powered end-devices.

The resumption scheme we describe reverses the roles of client (end-device) and server. At the same time it mitigates the issues related to memory space, computation cost, and amount of transmitted data.

7.4 Concrete Instantiation

In this section we present a concrete instantiation of our 3-AKE protocol described in Section 7.2. We have to choose a 2-AKE-secure protocol P, a 2-AKE-secure protocol P', and a secure

authenticated encryption function ENC.

Regarding protocol P, we recall that it must fulfill the properties listed in Section 7.2.3, which includes the essential forward secrecy. We describe an instantiation of P with (Section 7.4.2) and without (Section 7.4.1) the session resumption procedure for low-resource ED.

7.4.1 Forward Secret 2-AKE Protocol P

We instantiate the 2-AKE protocol P with the SAKE protocol described in Chapter 6. SAKE fulfills all the properties listed in Section 7.2.3.[1]

We recall that SAKE uses a key-evolving scheme, based on a one-way function, to update the symmetric root key shared by the two peers. Thus, applying SAKE, ED and KS compute an intermediary ik with their shared master key mk (used as SAKE root key). The current master key is then updated with the one-way function update: $mk^{i+1} \leftarrow \mathsf{update}(mk^i)$. Likewise, applying SAKE, ED and XS $\in \{\text{CS}, \text{AS}\}$ compute a session key sk with the key ik they share (used as SAKE root key in that case). Eventually, the SAKE root key used in that case is updated: $ik^{t+1} \leftarrow \mathsf{update}(ik^t)$.

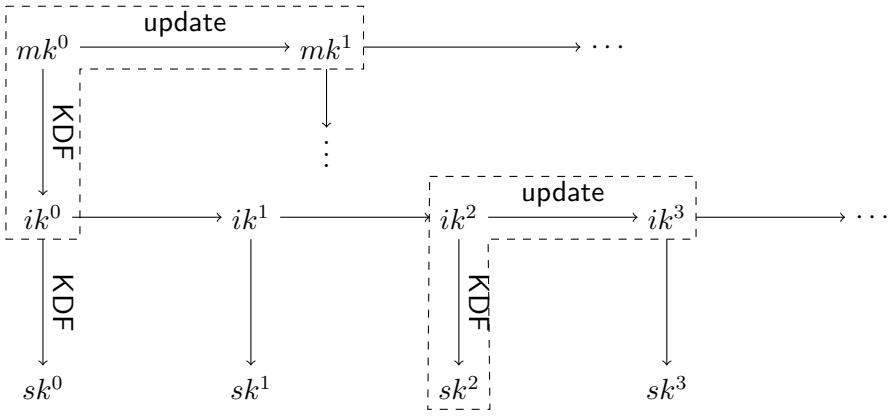


Figure 7.5 – Key chains in SAKE

Figure 7.5 depicts the evolution of the three types of keys over time: the master key mk, the intermediary key ik, and the session key sk. The computation of ik^0 and mk^1 from mk^0 corresponds to the 2-AKE run executed between ED and KS as depicted by Figure 7.2a. The computation of sk^2 and ik^3 from ik^2 corresponds to the 2-AKE run done between ED and CS (resp. AS) with $ik = ik_c$ (resp. $ik = ik_a$) as depicted by Figure 7.3a (resp. Figure 7.3b). Note that the keys ik_c and ik_a are computed from two different values mk (i.e., yielded by two different 2-AKE runs between ED and KS). In Figure 7.5, the branch $mk^0 \rightarrow mk^1 \rightarrow \cdots$ corresponds to the evolution of mk throughout successive key exchange runs executed between ED and KS. Each of these runs yields a new intermediary key $ik = ik^0$. The branch $ik^0 \rightarrow ik^1 \rightarrow \cdots$ corresponds to the evolution of ik throughout successive key exchange runs (each one outputting a session key sk^i) executed between ED and XS *without* the involvement of KS.

[1] Any other 2-AKE protocol can be used, as long as it provides the same properties as SAKE, but we are not aware of other such protocols.

7.4.2 Protocol P with Session Resumption Scheme for Low-resource ED

In this section, we describe a session resumption scheme dedicated to low-resource ED. This scheme (i) fulfills the features of the procedure described in Section 7.3.2, and (ii) is a 2-AKE-secure protocol (which include, in particular, forward secrecy). Recall that the session resumption procedure is made of two phases (see Section 7.3.2): the *storage phase* (phase (a)) and the *retrieval phase* (phase (b)). The retrieval phase of the scheme for low-resource ED is a variant of the SAKE protocol adapted to include the use of the ticket. We call this variant *SAKE-R*.

Session resumption procedure with SAKE-R. Figure 7.6 depicts the two phases of the procedure regarding the evolution of the keys.

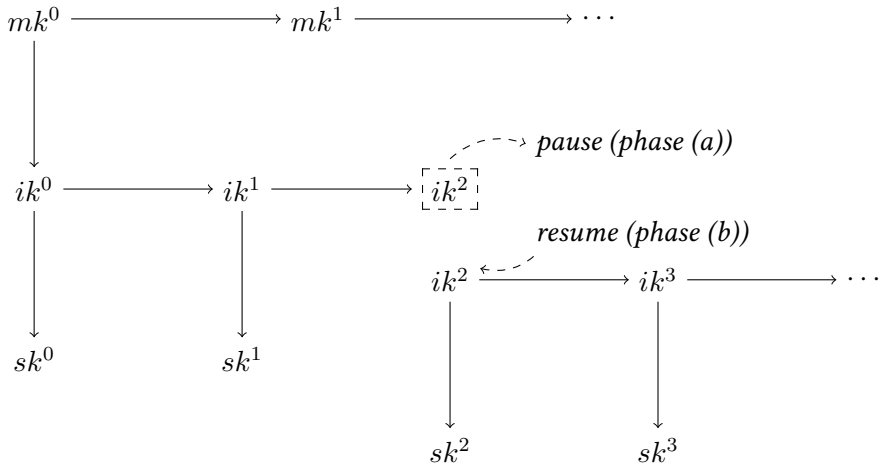


Figure 7.6 – Resuming a chain of intermediary keys (from ik^2)

Figure 7.7 depicts the storage phase of the session resumption procedure. The computation $\mathsf{H}^n(k_j) = \mathsf{H}(\cdots \mathsf{H}(\mathsf{H}(k_j)) \cdots)$ corresponds to n times the application of function H, where n is the "distance" between the current key k_j stored by ED and the (new) key k_i needed to encrypt ik (i.e., $n = i - j$). SAKE (hence SAKE-R) uses two main keys: an authentication key K' and a derivation key K. Therefore, ik corresponds to $K \| K'$.

The parameter id_{ticket} indicates what key (i.e., its index in the key chain) must be used to decrypt the ticket, and id_{server} identifies the server that stores the ticket.

During the storage phase, ED merely sends the ticket through the ongoing secure channel established with XS (see Section 7.3.2). When ED initiates the retrieval phase with SAKE-R (see Figure 7.8), the first message sent to XS carries an identifier of the ticket to retrieve. XS responds with the corresponding ticket. The parameter id_{ticket} indicates which key k_i (i.e., essentially its index i) must be used to decrypt the corresponding ticket.

The subsequent messages are essentially the same as the original SAKE protocol. They embed pseudo-random values that participate in the mutual authentication, and the session key computation. When the server is the initiator, the ticket is sent in the first message (see Figure 7.9).

Once ED gets ik, the ticket decryption key k_j currently kept by ED is replaced with $k_{i+1} = \mathsf{H}(k_i)$, where k_i is the key used to decrypt the retrieved ticket (see Section 7.3.2). Therefore, ED rejects any replay of an already consumed ticket.

We can observe that ED updates its root key $ik = K \| K'$ upon reception of message m_B. If XS does not receive message τ_A, it does not update its own root key ik. Hence ED and the

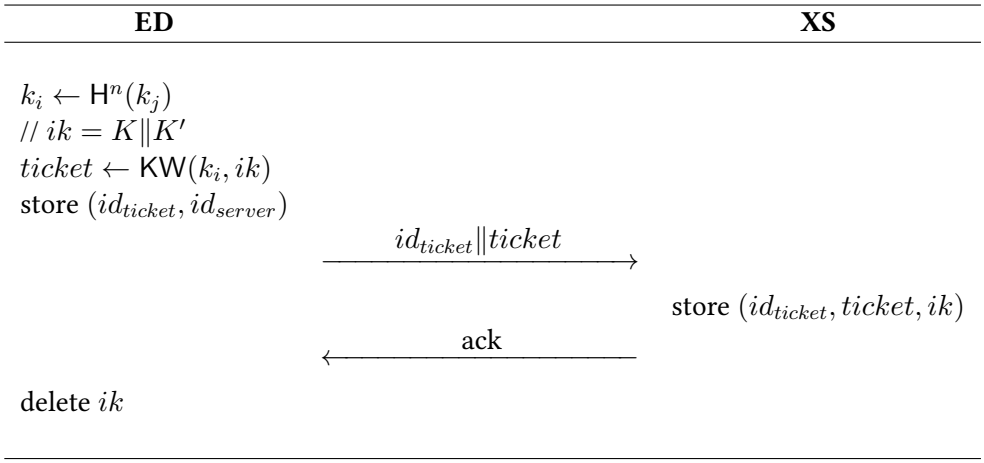


Figure 7.7 – Storage of a ticket

server are "desynchronised" (i.e., they do not share the same value of ik). When ED initiates anew the key exchange, it executes the SAKE protocol (and not SAKE-R) since it has already retrieved ik. SAKE enables ED and XS to resynchronise in the *continuity* of the protocol run (i.e., SAKE is *self-synchronising*). Therefore, ED and XS can perform a correct key exchange, and eventually compute a shared session key sk.

For the sake of clarity we use the following notation:

- kdf corresponds to: $sk \leftarrow \mathsf{KDF}(K, f(r_A, r_B))$
- upd corresponds to
  1. $K \leftarrow \mathsf{update}(K)$
  2. $K' \leftarrow \mathsf{update}(K')$

Function f is deliberately left undefined. For instance, $f(r_A, r_B)$ can be equal to the concatenation or the bitwise addition of r_A and r_B.[2] KDF is the session key derivation function used in SAKE (and SAKE-R). update is the one-way function used to update the root key (i.e., the intermediary key ik in that case). $\mathsf{Vrf}(k, m, \tau)$ denotes the MAC verification function. It takes as input a secret key k, a message m, and a tag τ. It outputs `true` if τ is a valid tag on message m with respect to k. Otherwise, it returns `false`.

We chose to model H and KDF as PRFs, and KW as an AE function.

7.4.3 Protocol P' and Function ENC

We instantiate the 2-AKE protocol P' with TLS 1.3. In order not to impair the security, we enforce (EC)DHE and forbid 0-RTT mode. We define the ENC function to be the record layer of TLS 1.3.

[2] The function f must be chosen such that the security of KDF is not impaired. We assume here that the cryptographic functions used are ideal (investigating this topic is beyond the scope of this chapter).

ED [A]		**XS [B]**
$(-)$		(K, K')
(k_j, id_{ticket})		$(id_{ticket}, ticket)$

$r_A \xleftarrow{\$} \{0,1\}^\lambda$
$m_A \leftarrow A \| r_A \| id_{ticket}$

$\xrightarrow{m_A}$

`if` (id_{ticket} not found) `then`
  abort

$r_B \xleftarrow{\$} \{0,1\}^\lambda$
$\tau_B \leftarrow \mathsf{MAC}(K', B \| A \| r_B \| r_A \| id_{ticket} \| ticket)$
$m_B \leftarrow r_B \| ticket \| \tau_B$

$\xleftarrow{m_B}$

$k_i \leftarrow \mathsf{H}^n(k_j)$
$ik \leftarrow \mathsf{KW}^{-1}(k_i, ticket)$
`if` $(ik = \perp)$ `then`
  abort

// $ik = K \| K'$
`if` $(\mathsf{Vrf}(K', B \| A \| r_B \| r_A \| id_{ticket} \| ticket, \tau_B) =$ `false`$)$ `then`
  abort

$k_{i+1} \leftarrow \mathsf{H}(k_i)$
delete k_j, k_i

$\tau_A \leftarrow \mathsf{MAC}(K', A \| B \| r_A \| r_B)$

$sk \leftarrow \mathsf{kdf}$
$K, K' \leftarrow \mathsf{upd}$

$\xrightarrow{\tau_A}$

`if` $(\mathsf{Vrf}(K', A \| B \| r_A \| r_B, \tau_A) =$ `false`$)$ `then`
  abort

$sk \leftarrow \mathsf{kdf}$
$K, K' \leftarrow \mathsf{upd}$

$\tau'_B \leftarrow \mathsf{MAC}(K', r_B \| r_A)$
delete $ticket$

$\xleftarrow{\tau'_B}$

`if` $(\mathsf{Vrf}(K', r_B \| r_A, \tau'_B) =$ `false`$)$ `then`
  abort

Figure 7.8 – Session Resumption with SAKE-R initiated by ED

ED [B]		**XS [A]**
$(-)$		(K, K')
(k_j, id_{ticket})		$(id_{ticket}, ticket)$
		$r_A \xleftarrow{\$} \{0,1\}^\lambda$
		$m_A \leftarrow A \Vert r_A \Vert id_{ticket} \Vert ticket$
	$\xleftarrow{m_A}$	
`if` (id_{ticket} not found) `then`		
abort		
$k_i \leftarrow \mathsf{H}^n(k_j)$		
$ik \leftarrow \mathsf{KW}^{-1}(k_i, ticket)$		
`if` $(ik = \perp)$ `then`		
abort		
$k_{i+1} \leftarrow \mathsf{H}(k_i)$		
delete k_j, k_i		
// $ik = K \Vert K'$		
$r_B \xleftarrow{\$} \{0,1\}^\lambda$		
$\tau_B \leftarrow \mathsf{MAC}(K', B \Vert A \Vert r_B \Vert r_A)$		
$m_B \leftarrow r_B \Vert \tau_B$		
$sk \leftarrow \mathsf{kdf}$		
$K, K' \leftarrow \mathsf{upd}$		
	$\xrightarrow{m_B}$	
		`if` $(\mathsf{Vrf}(K', B \Vert A \Vert r_B \Vert r_A, \tau_B) = \texttt{false})$ `then`
		abort
		$sk \leftarrow \mathsf{kdf}$
		$K, K' \leftarrow \mathsf{upd}$
		$\tau_A \leftarrow \mathsf{MAC}(K', A \Vert B \Vert r_A \Vert r_B)$
		delete $ticket$
	$\xleftarrow{\tau_A}$	
`if` $(\mathsf{Vrf}(K', A \Vert B \Vert r_A \Vert r_B, \tau_A) = \texttt{false})$ `then`		
abort		

Figure 7.9 – Session Resumption with SAKE-R initiated by XS $\in \{\text{CS}, \text{AS}\}$

7.5 Security Proofs

In this section we use the 3-AKE security model described in Chapter 5, Section 5.2, to prove the security of our schemes. We begin with the generic 3-AKE protocol Π depicted in Section 7.2, and follow with the instantiation described in Section 7.4.

7.5.1 Generic 3-AKE Protocol

The protocol Π is based on: (i) the 2-AKE protocol P executed between ED and KS, ED and XS, (ii) the 2-AKE protocol P' executed between KS and XS, and (iii) the function ENC used to set up a secure channel between KS and XS with a session key output by P'. Informally, the security of the 3-AKE protocol Π relies on the 2-AKE-security of P and P' (Chapter 2, Section 2.3.1), and on the sAE-security of the function ENC (with respect to the real-or-random variant, see Chapter 2, Section 2.2.5). Based on the security of P, P', and ENC, we show that Π is a secure 3-AKE protocol according to Definition 5.4.

P is a symmetric-key based protocol, whereas P' can implement asymmetric schemes. Therefore, with respect to the security model, we define the long-term key ltk of each party to be $\mathsf{ltk} = (\mathsf{pubk}, \mathsf{prvk}, \mathsf{rootk})$ where (i) pubk is a certified public key, (ii) prvk is the corresponding private key, and (iii) rootk is a symmetric root key. We recall that $\mathcal{K}$, $\mathcal{X}$, and $\mathcal{E}$ are respectively the sets of KS, XS, and ED parties. For any party in $\mathcal{K}$, the three components of ltk are defined. For any party in $\mathcal{X}$, ltk is fully defined *after* the first 3-AKE run (before, $\mathsf{rootk} = \perp$). For any party in $\mathcal{E}$, $\mathsf{ltk} = (\perp, \perp, \mathsf{rootk})$.

Theorem 7.1. *The protocol Π is a secure 3-AKE protocol under the assumption that P is a secure 2-AKE protocol, P' is a secure 2-AKE protocol, and ENC is a secure sAE function, and for any probabilistic polynomial time adversary $\mathcal{A}$ in the 3-AKE security experiment against Π*

$$\begin{aligned}
\mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) &= n_K \cdot n_E \cdot n_X \Big[2\mathsf{adv}_{P}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_1) + \mathsf{adv}_{P'}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_0) + \mathsf{adv}_{P'}^{\mathsf{key\text{-}ind}}(\mathcal{B}_0) \\
&\qquad + 2\mathsf{adv}_{\mathsf{ENC}}^{\mathsf{sae}}(\mathcal{B}_2)\Big] \\
\mathsf{adv}_{\Pi}^{\mathsf{key\text{-}ind}}(\mathcal{A}) &= \mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) + n_K \cdot n_E \cdot n_X \Big[2\mathsf{adv}_{P}^{\mathsf{key\text{-}ind}}(\mathcal{B}_1) + \mathsf{adv}_{P'}^{\mathsf{key\text{-}ind}}(\mathcal{B}_0) \\
&\qquad + \mathsf{adv}_{\mathsf{ENC}}^{\mathsf{sae}}(\mathcal{B}_2)\Big]
\end{aligned}$$

where n_K, n_E, n_X are respectively the number of KS, ED, and XS parties, and $\mathcal{B}_0$ is an adversary against the 2-AKE-security of P', $\mathcal{B}_1$ an adversary against the 2-AKE-security of P, and $\mathcal{B}_2$ an adversary against the sAE-security of ENC.

In order to prove Theorem 7.1, we proceed through a sequence of games [Sho04; BR04] between a challenger and an adversary $\mathcal{A}$.

Entity authentication. First we consider the entity authentication experiment described in Section 5.2.2.

Proof. Let E_i be the event that the adversary succeeds in making an instance accept maliciously in Game i. We use the following hops.

Game 0. This game corresponds to the entity authentication security experiment described in Section 5.2.2. Therefore we have that

$$\Pr[E_0] = \mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A}).$$

Game 1. In this game, the challenger aborts the experiment if it does not guess the party the adversary targets, and its party partners. There are respectively n_K, n_E, n_X parties in the sets $\mathcal{K}$, $\mathcal{E}$, and $\mathcal{X}$. Therefore we have that

$$\Pr[E_1] = \Pr[E_0] \times \frac{1}{n_K \cdot n_E \cdot n_X}.$$

Game 2. Now the parties $P_k \in \mathcal{K}$, $P_j \in \mathcal{X}$ and $P_i \in \mathcal{E}$ are fixed. In this game, the challenger aborts the experiment if the adversary succeeds in impersonating P_j to P_k or conversely. We reduce this event to the 2-AKE-security (with respect to entity authentication) of the protocol P' applied between P_k and P_j. Therefore we have that

$$\Pr[E_1] \leq \Pr[E_2] + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P'}(\mathcal{B}_0)$$

where $\mathcal{B}_0$ is an adversary against the 2-AKE-security of P'.

This guarantees that to each instance $\pi^s_k \in P_k.\mathsf{Instances}$ there exists a unique instance $\pi^v_j \in P_j.\mathsf{Instances}$ such that $\pi^s_k.\mathsf{sid} = \pi^v_j.\mathsf{sid}$ (and conversely).

Game 3. In this game, the challenger aborts the experiment if the adversary succeeds in impersonating P_i to P_k or conversely. We reduce this event to the 2-AKE-security (with respect to entity authentication) of the protocol P applied between P_k and P_i. Therefore we have that

$$\Pr[E_2] \leq \Pr[E_3] + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P}(\mathcal{B}_1)$$

where $\mathcal{B}_1$ is an adversary against the 2-AKE-security of P.

This guarantees that to each instance $\pi^m_i \in P_i.\mathsf{Instances}$ there exists a unique instance $\pi^\ell_k \in P_k.\mathsf{Instances}$ such that $\pi^m_i.\mathsf{sid} = \pi^\ell_k.\mathsf{sid}$ (and conversely).

Game 4. Another way for the adversary to win the entity authentication security experiment is to get the intermediary key ik used by P_i and P_j to mutually authenticate, or to forge a valid message intended to P_j that carries an intermediary key chosen by the adversary. In this game, the challenger aborts the experiment if the adversary succeeds in either case. The secure channel between P_k and P_j is established with the function ENC keyed with the session key output by P'. We reduce each of the two events to the sAE-security of ENC. In turn, we must assume that the ENC key is random. The latter is reduced to the 2-AKE-security (with respect to key indistinguishability) of P'. Therefore we have that

$$\Pr[E_3] \leq \Pr[E_4] + \mathsf{adv}^{\mathsf{key\text{-}ind}}_{P'}(\mathcal{B}_0) + 2\mathsf{adv}^{\mathsf{sae}}_{\mathsf{ENC}}(\mathcal{B}_2)$$

where $\mathcal{B}_2$ is an adversary against the sAE-security of ENC.

Game 5. In this game, the challenger aborts the experiment if the adversary succeeds in impersonating P_i to P_j or conversely. We reduce this event to the 2-AKE-security (with respect to entity authentication) of P. Therefore we have that

$$\Pr[E_4] \leq \Pr[E_5] + \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{P}(\mathcal{B}_1).$$

Due to Game 4 and Game 5, we have that, to each instance $\pi^n_i \in P_i.\mathsf{Instances}$, there exists a unique instance $\pi^u_j \in P_j.\mathsf{Instances}$ such that $\pi^n_i.\mathsf{sid} = \pi^u_j.\mathsf{sid}$ (and conversely). Due to Game 4, the intermediary key $\mathsf{ck} = ik$ shared by π^m_i and π^ℓ_k is also known to π^v_j (i.e., $ik = \pi^m_i.\mathsf{ck} = \pi^v_j.\mathsf{km}$). Due to Game 5, $\pi^n_i.\mathsf{sid} = \pi^u_j.\mathsf{sid}$. Hence $\pi^n_i.\mathsf{trscrpt} = \pi^u_j.\mathsf{trscrpt}$. We define function

g to be the key derivation function that outputs the session key $\pi_i^n.\mathsf{ck} = \pi_j^u.\mathsf{ck} = sk$ from the root key ik and the messages exchanged between π_i^n and π_j^u. Therefore, we have that $g(\pi_i^m.\mathsf{ck}, \pi_i^n.\mathsf{trscrpt}) = \pi_i^n.\mathsf{ck} = \pi_j^u.\mathsf{ck} = g(\pi_j^v.\mathsf{km}, \pi_j^u.\mathsf{trscrpt})$.

To that point, the adversary has no chance to win. Therefore

$$\Pr[E_5] = 0.$$

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

$$\begin{aligned}
\mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) &= \Pr[E_0] \\
&= n_K \cdot n_E \cdot n_X \cdot \Pr[E_1] \\
&\leq n_K \cdot n_E \cdot n_X \left[\Pr[E_2] + \mathsf{adv}_{P'}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_0)\right] \\
&\leq n_K \cdot n_E \cdot n_X \left[\Pr[E_3] + \mathsf{adv}_{P}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_1) + \mathsf{adv}_{P'}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_0)\right] \\
&\leq n_K \cdot n_E \cdot n_X \Big[\Pr[E_4] + \mathsf{adv}_{P'}^{\mathsf{key\text{-}ind}}(\mathcal{B}_0) + 2\mathsf{adv}_{\mathsf{ENC}}^{\mathsf{sae}}(\mathcal{B}_2) + \mathsf{adv}_{P}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_1) \\
&\qquad + \mathsf{adv}_{P'}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_0)\Big] \\
&\leq n_K \cdot n_E \cdot n_X \Big[\Pr[E_5] + \mathsf{adv}_{P'}^{\mathsf{key\text{-}ind}}(\mathcal{B}_0) + 2\mathsf{adv}_{\mathsf{ENC}}^{\mathsf{sae}}(\mathcal{B}_2) + 2\mathsf{adv}_{P}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_1) \\
&\qquad + \mathsf{adv}_{P'}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_0)\Big] \\
&\leq n_K \cdot n_E \cdot n_X \Big[\mathsf{adv}_{P'}^{\mathsf{key\text{-}ind}}(\mathcal{B}_0) + \mathsf{adv}_{P'}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_0) + 2\mathsf{adv}_{P}^{\mathsf{ent\text{-}auth}}(\mathcal{B}_1) \\
&\qquad + 2\mathsf{adv}_{\mathsf{ENC}}^{\mathsf{sae}}(\mathcal{B}_2)\Big]
\end{aligned}$$

□

Key indistinguishability. Now we consider the key indistinguishability security experiment described in Section 5.2.2.

In order to win the key indistinguishability experiment, the adversary can first try to break protocol P applied between ED and KS, or protocol P' applied between KS and XS. Then the adversary can target the session key shared between ED and XS through two ways. Firstly, the adversary can try to get the key ik sent by KS to XS, and protected with function ENC (ik is used by ED and XS to derive their session key). Secondly, the adversary can try to "directly" break protocol P applied between ED and XS which yields the session key. This is reflected by the successive games below.

Proof. Let E_i be the event that the adversary wins in Game i, and $\mathsf{adv}_i = \Pr[E_i] - \frac{1}{2}$. We use the following hops.

Game 0. This game corresponds to the key indistinguishability security experiment described in Section 5.2.2. Therefore we have that

$$\Pr[E_0] = \mathsf{adv}_0 + \frac{1}{2} = \mathsf{adv}_{\Pi}^{\mathsf{key\text{-}ind}}(\mathcal{A}) + \frac{1}{2}.$$

Game 1. In this game, the challenger aborts the experiment and chooses $b' \in \{0, 1\}$ uniformly at random if there exists an instance that maliciously accepts. In other words, we make the same modifications as in the games performed during the entity authentication proof. Therefore

$$\mathsf{adv}_0 \leq \mathsf{adv}_1 + \mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A}).$$

Game 2. In this game, the adversary aborts the experiment if it does not guess the party the adversary targets, and its party partners. There are respectively n_K, n_E, n_X parties in the sets $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{X}$. Therefore

$$\mathsf{adv}_2 = \mathsf{adv}_1 \times \frac{1}{n_K \cdot n_E \cdot n_X}.$$

Game 3. Now the parties $P_k \in \mathcal{K}$, $P_j \in \mathcal{X}$ and $P_i \in \mathcal{E}$ are fixed. Moreover, the conditions of Definition 5.2 are satisfied. This means in particular that each instance ending in accepting state is pairwise partnered with a unique instance.

In this game, we replace the session key ik output by the 2-AKE run of protocol P between P_i and P_k with a truly random value $\tilde{ik}$. The challenger aborts the game if there is an algorithm able to distinguish between both values. We reduce this event to the 2-AKE-security (with respect to key indistinguishability) of P. Therefore, we have that

$$\mathsf{adv}_2 \leq \mathsf{adv}_3 + \mathsf{adv}_P^{\mathsf{key\text{-}ind}}(\mathcal{B}_1)$$

where $\mathcal{B}_1$ is an adversary against the 2-AKE-security of P.

Game 4. In this game, we replace the session key output by the 2-AKE run of protocol P' between P_k and P_j with a truly random value. The challenger aborts the game if there is an algorithm able to distinguish between both values. We reduce this event to the 2-AKE-security (with respect to key indistinguishability) of P'. Therefore, we have that

$$\mathsf{adv}_3 \leq \mathsf{adv}_4 + \mathsf{adv}_{P'}^{\mathsf{key\text{-}ind}}(\mathcal{B}_0)$$

where $\mathcal{B}_0$ is an adversary against the 2-AKE-security of P'.

Game 5. The key (allegedly ik) sent by P_k to P_j is protected with ENC, which is keyed with the session key output by P'. The adversary can try to get the key $\tilde{ik}$, and then compute the session key sk shared between P_i and P_j. We reduce this event to the sAE-security of ENC. In turn, this relies implicitly on the fact that the encryption key used to key ENC (and which is output by P') be indistinguishable from random. This is the case due to Game 4. Therefore, in this game, the challenger aborts the experiment if the adversary succeeds in getting ik. We have that

$$\mathsf{adv}_4 \leq \mathsf{adv}_5 + \mathsf{adv}_{\mathsf{ENC}}^{\mathsf{sae}}(\mathcal{B}_2).$$

Game 6. Finally, in order to win the experiment, the adversary can try to break the 2-AKE-security (with respect to key indistinguishability) of P executed between P_i and P_j. Therefore we have that

$$\mathsf{adv}_5 \leq \mathsf{adv}_6 + \mathsf{adv}_P^{\mathsf{key\text{-}ind}}(\mathcal{B}_1).$$

To that point, the adversary can do no better than guessing. Therefore

$$\mathsf{adv}_6 = 0.$$

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

$$\begin{aligned}
\mathsf{adv}_{\Pi}^{\mathsf{key\text{-}ind}}(\mathcal{A}) &= \mathsf{adv}_0 \\
&\leq \mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) + \mathsf{adv}_1 \\
&\leq \mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) + n_K \cdot n_E \cdot n_X \cdot \mathsf{adv}_2 \\
&\leq \mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) + n_K \cdot n_E \cdot n_X \left[\mathsf{adv}_3 + \mathsf{adv}_P^{\mathsf{key\text{-}ind}}(\mathcal{B}_1)\right] \\
&\leq \mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) + n_K \cdot n_E \cdot n_X \left[\mathsf{adv}_4 + \mathsf{adv}_{P'}^{\mathsf{key\text{-}ind}}(\mathcal{B}_0) + \mathsf{adv}_P^{\mathsf{key\text{-}ind}}(\mathcal{B}_1)\right] \\
&\leq \mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) + n_K \cdot n_E \cdot n_X \Big[\mathsf{adv}_5 + \mathsf{adv}_{\mathsf{ENC}}^{\mathsf{sae}}(\mathcal{B}_2) + \mathsf{adv}_{P'}^{\mathsf{key\text{-}ind}}(\mathcal{B}_0) \\
&\qquad + \mathsf{adv}_P^{\mathsf{key\text{-}ind}}(\mathcal{B}_1)\Big] \\
&\leq \mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) + n_K \cdot n_E \cdot n_X \Big[\mathsf{adv}_6 + \mathsf{adv}_{\mathsf{ENC}}^{\mathsf{sae}}(\mathcal{B}_2) + \mathsf{adv}_{P'}^{\mathsf{key\text{-}ind}}(\mathcal{B}_0) \\
&\qquad + 2\mathsf{adv}_P^{\mathsf{key\text{-}ind}}(\mathcal{B}_1)\Big] \\
&\leq \mathsf{adv}_{\Pi}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) + n_K \cdot n_E \cdot n_X \Big[\mathsf{adv}_{\mathsf{ENC}}^{\mathsf{sae}}(\mathcal{B}_2) + \mathsf{adv}_{P'}^{\mathsf{key\text{-}ind}}(\mathcal{B}_0) \\
&\qquad + 2\mathsf{adv}_P^{\mathsf{key\text{-}ind}}(\mathcal{B}_1)\Big]
\end{aligned}$$

□

7.5.2 SAKE-R

In this section, we prove the security of SAKE-R. We consider the case where ED initiates SAKE-R (see Figure 7.8). The converse case, where the XS is the initiator, follows the same reasoning and is not detailed here.

With the following theorem we claim that SAKE-R is a secure 2-AKE protocol with respect to the security model of Brzuska et al. [BJS16] described in Chapter 2, Section 2.3.1. For any ED, the long-term key ltk is defined as $\mathsf{ltk} = (K, K', k_j)$. That is, a Corrupt-query returns the derivation master key K, the authentication master key K, and the ticket encryption key k_j. For any XS, ltk is defined as $\mathsf{ltk} = (K, K')$. In that case, a Corrupt-query returns K and K'.

Theorem 7.2. *The protocol SAKE-R is a secure 2-AKE protocol, and for any probabilistic polynomial time adversary $\mathcal{A}$ in the 2-AKE security experiment against SAKE-R*

$$\begin{aligned}
\mathsf{adv}_{SAKE\text{-}R}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) &\leq nq \Big[(nq-1)2^{-(\lambda-1)} + 2(q-1)\mathsf{adv}_{\mathsf{H}}^{\mathsf{prf}}(\mathcal{B}_0) + 2\mathsf{adv}_{\mathsf{KW}}^{\mathsf{ae}}(\mathcal{B}_1) \\
&\qquad + 3\mathsf{adv}_{\mathsf{Tag}}^{\mathsf{suf\text{-}cma}}(\mathcal{B}_2) + q \cdot \mathsf{adv}_{\mathsf{update}}^{\mathsf{prf}}(\mathcal{B}_3)\Big] \\
\mathsf{adv}_{SAKE\text{-}R}^{\mathsf{key\text{-}ind}}(\mathcal{A}) &\leq \mathsf{adv}_{SAKE\text{-}R}^{\mathsf{ent\text{-}auth}}(\mathcal{A}) + nq \Big[2\left(\mathsf{adv}_{\mathsf{KW}}^{\mathsf{ae}}(\mathcal{B}_1) + \mathsf{adv}_{\mathsf{KDF}}^{\mathsf{prf}}(\mathcal{B}_4)\right) \\
&\qquad + (q-1)\left(\mathsf{adv}_{\mathsf{update}}^{\mathsf{prf}}(\mathcal{B}_3) + 2\mathsf{adv}_{\mathsf{H}}^{\mathsf{prf}}(\mathcal{B}_0)\right)\Big]
\end{aligned}$$

where n is the number of parties (ED and XS), q the maximum number of instances (sessions) per party, λ the size of the pseudo-random values (r_A, r_B), $\mathcal{B}_0$ an adversary against the PRF-security of H, $\mathcal{B}_1$ an adversary against the AE-security of KW, $\mathcal{B}_2$ an adversary against the SUF-CMA-security of $\mathsf{Tag} = (\mathsf{Tag.Gen}, \mathsf{Tag.MAC}, \mathsf{Tag.Vrf})$, $\mathcal{B}_3$ an adversary against the PRF-security of update, and $\mathcal{B}_4$ an adversary against the PRF-security of KDF.

The security proof for SAKE-R follows essentially the same steps as the proof for SAKE (see Chapter 6, Section 6.3).

We define functions H and update to be two PRFs. That is $\mathsf{H} : y \mapsto \mathsf{PRF}_{\mathsf{H}}(y, x)$ and $\mathsf{update} : y \mapsto \mathsf{PRF}_{\mathsf{update}}(y, x')$ for some (constant) values x and x'.

Entity authentication. We start with the 2-AKE entity authentication experiment.

Proof. Let $\mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R}(\mathcal{A})$ be the probability that the adversary wins the entity authentication game. Let $\mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R,\mathsf{client}}(\mathcal{A})$ be the probability that an adversary succeeds against a client (ED), and $\mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R,\mathsf{server}}(\mathcal{A})$ the probability that an adversary succeeds against a server (XS). We have that

$$\mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R}(\mathcal{A}) \leq \mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R,\mathsf{client}}(\mathcal{A}) + \mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R,\mathsf{server}}(\mathcal{A}).$$

Client adversary. We first consider an adversary that targets a client. Let E_i be the event that the adversary succeeds in making a client instance accept maliciously in Game$^{\mathsf{client}}$ i.

Game$^{\mathsf{client}}$ 0. This game corresponds to the 2-AKE entity authentication security experiment when the adversary targets a client instance. Therefore

$$\Pr[E_0] = \mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R,\mathsf{client}}(\mathcal{A}).$$

Game$^{\mathsf{client}}$ 1. In this game, the challenger aborts the experiment if there exists an instance that chooses a random value r_A or r_B that is not unique. There is at most $n \times q$ random values, each uniformly drawn at random in $\{0,1\}^\lambda$. Hence the two games are equivalent up to a collision term $\frac{nq(nq-1)}{2^\lambda}$. Therefore

$$\Pr[E_0] \leq \Pr[E_1] + \frac{nq(nq-1)}{2^\lambda}.$$

Game$^{\mathsf{client}}$ 2. In this game, the challenger aborts the experiment if it does not guess which client instance will be the first to maliciously accept. There is n parties and q instances per party. Therefore we have that

$$\Pr[E_2] = \Pr[E_1] \times \frac{1}{nq}.$$

Game$^{\mathsf{client}}$ 3. The first key k_0 used to compute a ticket is uniformly drawn at random. The next encryption key k_1 is computed as $k_1 = \mathsf{H}(k_0) = \mathsf{PRF}_{\mathsf{H}}(k_0, x)$. Since k_0 is random, we can replace $\mathsf{PRF}_{\mathsf{H}}(k_0, \cdot)$ with a truly random function $\mathsf{F}^{\mathsf{H}}_{k_0}$. We do the same for any server instance that uses function H with the same key k_0 to compute k_1. Distinguishing the change implies an algorithm able to distinguish function H from a random function. This corresponds to an advantage $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0)$ where $\mathcal{B}_0$ is an adversary against the PRF-security of H. Since $\mathsf{PRF}_{\mathsf{H}}(k_0, \cdot)$ is replaced with a random function $\mathsf{F}^{\mathsf{H}}_{k_0}$, $k_1 = \mathsf{F}^{\mathsf{H}}_{k_0}(x)$ is random. In turn, we can replace $\mathsf{PRF}_{\mathsf{H}}(k_1, \cdot)$ with a truly random function $\mathsf{F}^{\mathsf{H}}_{k_1}$. Recursively, we replace each function $\mathsf{PRF}_{\mathsf{H}}(k_i, \cdot)$ with a truly random function $\mathsf{F}^{\mathsf{H}}_{k_i}$. There is at most q instances per party, hence at most $q-1$ updates of the original key k_0 before computing a ticket (that is, $0 \leq i < q$). Therefore, distinguishing the successive changes corresponds to an advantage at most $(q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0)$. Consequently, in this game, the challenger aborts the experiment if the adversary is able to distinguish any of these changes. Therefore, we have that

$$\Pr[E_2] \leq \Pr[E_3] + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0).$$

Game$^{\text{client}}$ 4. In this game, the challenger aborts the experiment if the adversary is able to get the key ik from $ticket = \mathsf{KW}(k_i, ik)$, $0 \leq i < q$. We reduce this event to the AE-security of function KW which guarantees real-from-random indistinguishability for the plaintexts (this is possible because k_i is indistinguishable from random due to Game$^{\text{client}}$ 3). Therefore we have that

$$\Pr[E_3] \leq \Pr[E_4] + \mathsf{adv}^{\mathsf{ae}}_{\mathsf{KW}}(\mathcal{B}_1)$$

where $\mathcal{B}_1$ is an adversary against the AE-security of KW.

Game$^{\text{client}}$ 5. In this game, the challenger aborts the experiment if the targeted instance π ever receives a valid message m_B but no instance partnered with π has output that message. Due to Game$^{\text{client}}$ 4, $ik = K \| K'$ (hence K') can be safely replaced with a truly random value. Therefore, we reduce this event to the SUF-CMA-security of the MAC function (keyed with K') used to compute m_B. Therefore, we have that

$$\Pr[E_4] \leq \Pr[E_5] + \mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Tag}}(\mathcal{B}_2)$$

where $\mathcal{B}_2$ is an adversary against the SUF-CMA-security of Tag.

Game$^{\text{client}}$ 6. The key used to compute the MAC tag τ'_B is $\mathsf{update}(K') = \mathsf{PRF}_{\mathsf{update}}(K', x')$. In this game, we replace $\mathsf{PRF}_{\mathsf{update}}(K', \cdot)$ with a random function $\mathsf{F}^{\mathsf{update}}_{K'}$. We do the same for any server instance that uses the update function with the same key K'. Distinguishing the change implies an algorithm able to distinguish the function update from a random function. Therefore, in this game, the challenger aborts the experiment if the adversary is able to distinguish such a change. Hence

$$\Pr[E_5] \leq \Pr[E_6] + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3)$$

where $\mathcal{B}_3$ is an adversary against the PRF-security of update.

Game$^{\text{client}}$ 7. In this game, the challenger aborts the experiment if the targeted instance π ever receives a valid message τ'_B but no instance partnered with π has output that message. Due to Game$^{\text{client}}$ 6, the key used to compute the MAC tag τ'_B is truly random. Hence, we reduce this event to the SUF-CMA-security of the MAC function used to compute τ'_B. Therefore, we have that

$$\Pr[E_6] \leq \Pr[E_7] + \mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Tag}}(\mathcal{B}_2).$$

To that point, the adversary has no chance to win. Therefore

$$\Pr[E_7] = 0.$$

Collecting all the probabilities from Game$^{\text{client}}$ 0 to Game$^{\text{client}}$ 7, we have that

$$\begin{aligned}
\mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R,\text{client}}(\mathcal{A}) &= \Pr[E_0] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + \Pr[E_1] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq \times \Pr[E_2] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\left[\Pr[E_3] + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0)\right] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\left[\Pr[E_4] + \mathsf{adv}^{\text{ae}}_{\mathsf{KW}}(\mathcal{B}_1) + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0)\right] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\left[\Pr[E_5] + \mathsf{adv}^{\text{suf-cma}}_{\mathsf{Tag}}(\mathcal{B}_2) + \mathsf{adv}^{\text{ae}}_{\mathsf{KW}}(\mathcal{B}_1)\right. \\
&\qquad \left. + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0)\right] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\left[\Pr[E_6] + \mathsf{adv}^{\text{prf}}_{\mathsf{update}}(\mathcal{B}_3) + \mathsf{adv}^{\text{suf-cma}}_{\mathsf{Tag}}(\mathcal{B}_2)\right. \\
&\qquad \left. + \mathsf{adv}^{\text{ae}}_{\mathsf{KW}}(\mathcal{B}_1) + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0)\right] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\left[\Pr[E_7] + \mathsf{adv}^{\text{prf}}_{\mathsf{update}}(\mathcal{B}_3) + 2\mathsf{adv}^{\text{suf-cma}}_{\mathsf{Tag}}(\mathcal{B}_2)\right. \\
&\qquad \left. + \mathsf{adv}^{\text{ae}}_{\mathsf{KW}}(\mathcal{B}_1) + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0)\right] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\left[\mathsf{adv}^{\text{prf}}_{\mathsf{update}}(\mathcal{B}_3) + 2\mathsf{adv}^{\text{suf-cma}}_{\mathsf{Tag}}(\mathcal{B}_2) + \mathsf{adv}^{\text{ae}}_{\mathsf{KW}}(\mathcal{B}_1)\right. \\
&\qquad \left. + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0)\right] \\
&\leq nq\left[(nq-1)2^{-\lambda} + \mathsf{adv}^{\text{prf}}_{\mathsf{update}}(\mathcal{B}_3) + 2\mathsf{adv}^{\text{suf-cma}}_{\mathsf{Tag}}(\mathcal{B}_2) + \mathsf{adv}^{\text{ae}}_{\mathsf{KW}}(\mathcal{B}_1)\right. \\
&\qquad \left. + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0)\right]
\end{aligned}$$

Server adversary. Now we consider an adversary that targets a server. Let E_i be the event that the adversary succeeds in making a server instance accept maliciously in Game$^{\text{server}}$ i.

Game$^{\text{server}}$ 0. This game corresponds to the 2-AKE entity authentication security experiment when the adversary targets a server instance. Therefore we have that

$$\Pr[E_0] = \mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R,\text{server}}(\mathcal{A}).$$

Game$^{\text{server}}$ 1. In this game, the challenger aborts the experiment if there exists an instance that chooses a random value r_A or r_B that is not unique. There is at most $n \times q$ random values, each uniformly drawn at random in $\{0,1\}^\lambda$. Hence the two games are equivalent up to a collision term $\frac{nq(nq-1)}{2^\lambda}$. Therefore

$$\Pr[E_0] \leq \Pr[E_1] + \frac{nq(nq-1)}{2^\lambda}.$$

Game$^{\text{server}}$ 2. In this game, the challenger aborts the experiment if it does not guess which server instance will be the first to maliciously accept. There is n parties and q instances per party. Therefore we have that

$$\Pr[E_2] = \Pr[E_1] \times \frac{1}{nq}.$$

Game$^{\text{server}}$ **3.** In this game, we apply the same changes as in Game$^{\text{client}}$ 3. That is, since $k_{i+1} = \mathsf{H}(k_i) = \mathsf{PRF}_{\mathsf{H}}(k_i, x)$, $i \geq 0$, we replace recursively $\mathsf{PRF}_{\mathsf{H}}(k_i, \cdot)$ with a truly random function $\mathsf{F}^{\mathsf{H}}_{k_i}$. Each change yields a loss $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0)$. Therefore, we have that

$$\Pr[E_2] \leq \Pr[E_3] + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0).$$

Game$^{\text{server}}$ **4.** In this game, the challenger aborts the experiment if the adversary is able to get the key ik from $ticket = \mathsf{KW}(k_i, ik)$, $0 \leq i < q$. We reduce this event to the AE-security of function KW (this is possible because k_i is indistinguishable from random due to Game$^{\text{server}}$ 3). Therefore we have that

$$\Pr[E_3] \leq \Pr[E_4] + \mathsf{adv}^{\mathsf{ae}}_{\mathsf{KW}}(\mathcal{B}_1)$$

where $\mathcal{B}_1$ is an adversary against the AE-security of KW.

Game$^{\text{server}}$ **5.** The first value of K' (K'_0) used to compute a MAC tag τ_A is uniformly chosen at random. During the next protocol run, the key is replaced with $\mathsf{update}(K'_0) = \mathsf{PRF}_{\mathsf{update}}(K'_0, x')$. Since K'_0 is random, we can replace $\mathsf{PRF}_{\mathsf{update}}(K'_0, \cdot)$ with a truly random function $\mathsf{F}^{\mathsf{update}}_{K'_0}$. We do the same for any client instance that uses function update with the same key K'_0. Distinguishing the change implies an algorithm able to distinguish the function update from a random function. This corresponds to an advantage $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3)$. Since $\mathsf{PRF}_{\mathsf{update}}(K'_0, \cdot)$ is replaced with a random function $\mathsf{F}^{\mathsf{update}}_{K'_0}$, $K'_1 = \mathsf{F}^{\mathsf{update}}_{K'_0}(x')$ is random. In turn, we can replace $\mathsf{PRF}_{\mathsf{update}}(K'_1, \cdot)$ with a truly random function $\mathsf{F}^{\mathsf{update}}_{K'_1}$. Recursively, we replace each function $\mathsf{PRF}_{\mathsf{update}}(K'_i, \cdot)$ with a truly random function $\mathsf{F}^{\mathsf{update}}_{K'_i}$. There is at most q instances per party, hence at most $q-1$ updates of the original key K'_0 before computing a MAC tag τ_A (that is, $0 \leq i < q$). Therefore, distinguishing the successive changes corresponds to an advantage at most $(q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3)$. Consequently, in this game, the challenger aborts the experiment if the adversary succeeds in distinguishing any of these changes. Therefore, we have that

$$\Pr[E_4] \leq \Pr[E_5] + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3).$$

Game$^{\text{server}}$ **6.** In this game, the challenger aborts the experiment if the targeted instance π ever receives a valid message τ_A but no instance partnered with π has output that message. Such a forgery can be achieved in one of two ways: either the adversary succeeds in forging a valid MAC tag τ_A, or it gets the key K' carried in $ticket$. We reduce the first possibility to the SUF-CMA-security of the MAC function used to compute τ_A. We reduce the second possibility to the AE-security of function KW, which is already assumed due to Game$^{\text{server}}$ 4. Therefore, we have that

$$\Pr[E_5] \leq \Pr[E_6] + \mathsf{adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{Tag}}(\mathcal{B}_2).$$

To that point, the adversary has no chance to win. Hence

$$\Pr[E_6] = 0.$$

Collecting all the probabilities from Game$^{\text{server}}$ 0 to Game$^{\text{server}}$ 6, we have that

$$\begin{aligned}
\mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R,\text{server}}(\mathcal{A}) &= \Pr[E_0] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + \Pr[E_1] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq \times \Pr[E_2] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\left[\Pr[E_3] + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0)\right] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\left[\Pr[E_4] + \mathsf{adv}^{\text{ae}}_{\mathsf{KW}}(\mathcal{B}_1) + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0)\right] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\Big[\Pr[E_5] + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{update}}(\mathcal{B}_3) + \mathsf{adv}^{\text{ae}}_{\mathsf{KW}}(\mathcal{B}_1) \\
&\qquad + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0)\Big] \\
&\leq \frac{nq(nq-1)}{2^\lambda} + nq\Big[\Pr[E_6] + \mathsf{adv}^{\text{suf-cma}}_{\mathsf{Tag}}(\mathcal{B}_2) \\
&\qquad + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{update}}(\mathcal{B}_3) + \mathsf{adv}^{\text{ae}}_{\mathsf{KW}}(\mathcal{B}_1) \\
&\qquad + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0)\Big] \\
&\leq nq\Big[(nq-1)2^{-\lambda} + \mathsf{adv}^{\text{suf-cma}}_{\mathsf{Tag}}(\mathcal{B}_2) + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{update}}(\mathcal{B}_3) \\
&\qquad + \mathsf{adv}^{\text{ae}}_{\mathsf{KW}}(\mathcal{B}_1) + (q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0)\Big]
\end{aligned}$$

Finally, we have that

$$\begin{aligned}
\mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R}(\mathcal{A}) &\leq \mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R,\text{client}}(\mathcal{A}) + \mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R,\text{server}}(\mathcal{A}) \\
&\leq nq\Big[(nq-1)2^{-(\lambda-1)} + 2(q-1)\mathsf{adv}^{\text{prf}}_{\mathsf{H}}(\mathcal{B}_0) + 2\mathsf{adv}^{\text{ae}}_{\mathsf{KW}}(\mathcal{B}_1) \\
&\qquad + 3\mathsf{adv}^{\text{suf-cma}}_{\mathsf{Tag}}(\mathcal{B}_2) + q \cdot \mathsf{adv}^{\text{prf}}_{\mathsf{update}}(\mathcal{B}_3)\Big]
\end{aligned}$$

□

Key indistinguishability. Now we consider the 2-AKE key indistinguishability security experiment.

Proof. Let E_i be the event that the adversary succeeds in making an instance accept maliciously in Game i, and $\mathsf{adv}_i = \Pr[E_i] - \frac{1}{2}$.

Game 0. This game corresponds to the 2-AKE key indistinguishability security experiment. Therefore we have

$$\Pr[E_0] = \frac{1}{2} + \mathsf{adv}^{\text{key-ind}}_{SAKE\text{-}R}(\mathcal{A}) = \frac{1}{2} + \mathsf{adv}_0.$$

Game 1. In this game, the challenger aborts the experiment and chooses $b \in \{0,1\}$ uniformly at random if there exists an instance that accepts maliciously. Therefore we have

$$\mathsf{adv}_0 \leq \mathsf{adv}_1 + \mathsf{adv}^{\text{ent-auth}}_{SAKE\text{-}R}.$$

Game 2. In this game, the challenger aborts the experiment if it does not guess which instance the adversary targets. Therefore, we have that

$$\mathsf{adv}_2 = \mathsf{adv}_1 \times \frac{1}{nq}.$$

Game 3. We distinguish two cases: the adversary targets either a client instance or a server instance, corresponding respectively to an advantage $\mathsf{adv}^{\text{key-ind}}_{SAKE\text{-}R,\mathsf{client}}(\mathcal{A})$ and $\mathsf{adv}^{\text{key-ind}}_{SAKE\text{-}R,\mathsf{server}}(\mathcal{A})$. Therefore we have that

$$\mathsf{adv}_2 \leq \mathsf{adv}^{\text{key-ind}}_{SAKE\text{-}R,\mathsf{client}}(\mathcal{A}) + \mathsf{adv}^{\text{key-ind}}_{SAKE\text{-}R,\mathsf{server}}(\mathcal{A}).$$

We begin with the first case.

Game$^{\mathsf{client}}$ 3. In this game, we apply the same changes as in Game$^{\mathsf{client}}$ 3 of the entity authentication experiment. That is, since $k_{i+1} = \mathsf{H}(k_i) = \mathsf{PRF}_{\mathsf{H}}(k_i, x)$, $i \geq 0$, we replace recursively $\mathsf{PRF}_{\mathsf{H}}(k_i, \cdot)$ with a truly random function $\mathsf{F}^{\mathsf{H}}_{k_i}$. Each change yields a loss $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0)$. Therefore, we have that

$$\mathsf{adv}^{\text{key-ind}}_{SAKE\text{-}R,\mathsf{client}}(\mathcal{A}) \leq \mathsf{adv}^{\mathsf{client}}_3 + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0).$$

Game$^{\mathsf{client}}$ 4. In this game, the challenger aborts the experiment if the adversary succeeds in getting K from $ticket = \mathsf{KW}(k_i, ik)$. We reduce this event to the AE-security of the KW function (we use the fact that k_i be indistinguishable from random due to Game$^{\mathsf{client}}$ 3). Therefore we have that

$$\mathsf{adv}^{\mathsf{client}}_3 \leq \mathsf{adv}^{\mathsf{client}}_4 + \mathsf{adv}^{\mathsf{ae}}_{\mathsf{KW}}(\mathcal{B}_1).$$

Game$^{\mathsf{client}}$ 5. In this game, we replace the KDF function used to compute the session key sk when keyed with K, with a random function $\mathsf{F}^{\mathsf{KDF}}_K$. We use the fact that K be indistinguishable from random due to Game$^{\mathsf{client}}$ 4. Consequently, the challenger aborts the experiment if the adversary succeeds in distinguishing the change. Therefore, we have that

$$\mathsf{adv}^{\mathsf{client}}_4 \leq \mathsf{adv}^{\mathsf{client}}_5 + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}(\mathcal{B}_4)$$

where $\mathcal{B}_4$ is an adversary against the PRF-security of KDF.

To that point, $sk = \mathsf{F}^{\mathsf{KDF}}_K(f(r_A, r_B))$ is a random value. Therefore the adversary can do no better than guessing. Hence

$$\mathsf{adv}^{\mathsf{client}}_5 = 0.$$

Collecting the probabilities from Game$^{\mathsf{client}}$ 3 to Game$^{\mathsf{client}}$ 5, we have that

$$\begin{aligned}
\mathsf{adv}^{\text{key-ind}}_{SAKE\text{-}R,\mathsf{client}}(\mathcal{A}) &\leq \mathsf{adv}^{\mathsf{client}}_3 + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0) \\
&\leq \mathsf{adv}^{\mathsf{client}}_4 + \mathsf{adv}^{\mathsf{ae}}_{\mathsf{KW}}(\mathcal{B}_1) + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0) \\
&\leq \mathsf{adv}^{\mathsf{client}}_5 + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}(\mathcal{B}_4) + \mathsf{adv}^{\mathsf{ae}}_{\mathsf{KW}}(\mathcal{B}_1) + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0) \\
&\leq \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}(\mathcal{B}_4) + \mathsf{adv}^{\mathsf{ae}}_{\mathsf{KW}}(\mathcal{B}_1) + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0)
\end{aligned}$$

Now we consider the case where the adversary targets a server instance.

Game$^{\text{server}}$ 3. The first value of K (K_0) used to compute the session key is uniformly chosen at random. During the next protocol run, the key is replaced with $\mathsf{update}(K_0) = \mathsf{PRF}_{\mathsf{update}}(K_0, x')$. Since K_0 is random, we can replace $\mathsf{PRF}_{\mathsf{update}}(K_0, \cdot)$ with a truly random function $\mathsf{F}^{\mathsf{update}}_{K_0}$. We do the same for any client instance that uses update function with the same key K_0. Distinguishing the change implies an algorithm able to distinguish the function update from a random function. This corresponds to an advantage $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3)$. Since $\mathsf{PRF}_{\mathsf{update}}(K_0, \cdot)$ is replaced with a random function $\mathsf{F}^{\mathsf{update}}_{K_0}$, $K_1 = \mathsf{F}^{\mathsf{update}}_{K_0}(x')$ is random. In turn, we can replace $\mathsf{PRF}_{\mathsf{update}}(K_1, \cdot)$ with a truly random function $\mathsf{F}^{\mathsf{update}}_{K_1}$. Recursively, we replace each function $\mathsf{PRF}_{\mathsf{update}}(K_i, \cdot)$ with a truly random function $\mathsf{F}^{\mathsf{update}}_{K_i}$. There is at most q instances per party, hence at most $q-1$ updates of the original key K_0 before computing a session key (that is, $0 \leq i < q$). Therefore, distinguishing the successive changes corresponds to an advantage at most $(q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3)$. Consequently, in this game, the challenger aborts the experiment if the adversary succeeds in distinguishing any of these changes. Therefore, we have that

$$\mathsf{adv}^{\mathsf{key\text{-}ind}}_{SAKE\text{-}R,\mathsf{server}}(\mathcal{A}) \leq \mathsf{adv}^{\mathsf{server}}_3 + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3).$$

Game$^{\text{server}}$ 4. In this game, we apply the same changes as in Game$^{\text{client}}$ 3 of the entity authentication experiment. That is, since $k_{i+1} = \mathsf{H}(k_i) = \mathsf{PRF}_{\mathsf{H}}(k_i, x)$, $i \geq 0$, we replace recursively $\mathsf{PRF}_{\mathsf{H}}(k_i, \cdot)$ with a truly random function $\mathsf{F}^{\mathsf{H}}_{k_i}$. Each change yields a loss $\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0)$. Therefore, we have that

$$\mathsf{adv}^{\mathsf{server}}_3 \leq \mathsf{adv}^{\mathsf{server}}_4 + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0).$$

Game$^{\text{server}}$ 5. In this game, the challenger aborts the experiment if the adversary succeeds in getting K from $ticket = \mathsf{KW}(k_i, ik)$. We reduce this event to the AE-security of the KW function (we use the fact that k_i be indistinguishable from random due to Game$^{\text{server}}$ 4). Therefore we have that

$$\mathsf{adv}^{\mathsf{server}}_4 \leq \mathsf{adv}^{\mathsf{server}}_5 + \mathsf{adv}^{\mathsf{ae}}_{\mathsf{KW}}(\mathcal{B}_1).$$

Game$^{\text{client}}$ 6. In this game, we replace the KDF function used to compute the session key sk when keyed with K, with a random function $\mathsf{F}^{\mathsf{KDF}}_K$. We use the fact that K be indistinguishable from random due to Game$^{\text{server}}$ 3 and Game$^{\text{server}}$ 5. Consequently, the challenger aborts the experiment if the adversary succeeds in distinguishing the change. Therefore, we have that

$$\mathsf{adv}^{\mathsf{server}}_5 \leq \mathsf{adv}^{\mathsf{server}}_6 + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}(\mathcal{B}_4).$$

To that point, $sk = \mathsf{F}^{\mathsf{KDF}}_K(f(r_A, r_B))$ is a random value. Therefore the adversary can do no better than guessing. Hence

$$\mathsf{adv}^{\mathsf{server}}_6 = 0.$$

Collecting the probabilities from Game$^{\text{server}}$ 3 to Game$^{\text{server}}$ 6, we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{key\text{-}ind}}_{SAKE\text{-}R,\mathsf{server}}(\mathcal{A}) &\leq \mathsf{adv}^{\mathsf{server}}_3 + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3) \\
&\leq \mathsf{adv}^{\mathsf{server}}_4 + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0) + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3) \\
&\leq \mathsf{adv}^{\mathsf{server}}_5 + \mathsf{adv}^{\mathsf{ae}}_{\mathsf{KW}}(\mathcal{B}_1) + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0) + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3) \\
&\leq \mathsf{adv}^{\mathsf{server}}_6 + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}(\mathcal{B}_4) + \mathsf{adv}^{\mathsf{ae}}_{\mathsf{KW}}(\mathcal{B}_1) + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0) \\
&\quad +(q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3) \\
&\leq \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}(\mathcal{B}_4) + \mathsf{adv}^{\mathsf{ae}}_{\mathsf{KW}}(\mathcal{B}_1) + (q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0) \\
&\quad +(q-1)\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3)
\end{aligned}$$

Finally, collecting all the probabilities, we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{key\text{-}ind}}_{SAKE\text{-}R}(\mathcal{A}) &= \mathsf{adv}_0 \\
&\leq \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE\text{-}R}(\mathcal{A}) + \mathsf{adv}_1 \\
&\leq \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE\text{-}R}(\mathcal{A}) + nq \cdot \mathsf{adv}_2 \\
&\leq \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE\text{-}R}(\mathcal{A}) + nq \left[\mathsf{adv}^{\mathsf{key\text{-}ind}}_{SAKE\text{-}R,\mathsf{client}}(\mathcal{A}) + \mathsf{adv}^{\mathsf{key\text{-}ind}}_{SAKE\text{-}R,\mathsf{server}}(\mathcal{A})\right] \\
&\leq \mathsf{adv}^{\mathsf{ent\text{-}auth}}_{SAKE\text{-}R}(\mathcal{A}) + nq \Big[(q-1)\left(\mathsf{adv}^{\mathsf{prf}}_{\mathsf{update}}(\mathcal{B}_3) + 2\mathsf{adv}^{\mathsf{prf}}_{\mathsf{H}}(\mathcal{B}_0)\right) \\
&\qquad\qquad +2\left(\mathsf{adv}^{\mathsf{ae}}_{\mathsf{KW}}(\mathcal{B}_1) + \mathsf{adv}^{\mathsf{prf}}_{\mathsf{KDF}}(\mathcal{B}_4)\right)\Big]
\end{aligned}$$

□

7.5.3 Achieving 3-AKE Security

In Chapter 6 we have proved that $P =$ SAKE is a secure 2-AKE protocol in the Brzuska et al. [BJS16] security model (which captures forward secrecy). Moreover $P =$ SAKE-R is a 2-AKE-secure protocol from Theorem 7.2. With respect to the 3-AKE security model, we define the long-term key component rootk of any ED to be $\mathsf{rootk} = (K, K')$ if $P =$ SAKE, and $\mathsf{rootk} = (K, K', k_j)$ if $P =$ SAKE-R. That is, in the latter case, we allow the 3-AKE adversary to get the ticket encryption key k_j through a $\mathsf{Corrupt}$-query, in addition to the derivation master key K and the authentication master key K'.

$P' =$ TLS 1.3 is proved to be a secure 2-AKE protocol [DFGS16]. Although this result applies to an earlier draft of the protocol, we may reasonably assume that the final version also guarantees 2-AKE-security.

ENC defined as the record layer of TLS 1.3 is proved to be AE-secure [BMM+15] in the sense of Rogaway [Rog02] (indistinguishability from random bits) which implies AE-security in the sense of Shrimpton [Shr04] (real-from-random indistinguishability). In addition, in TLS 1.3, a per-record nonce derived from a sequence number aims at guaranteeing non-replayability of the records (the sequence number being maintained independently at both sides). Hence we assume the sAE-security of ENC.

Hence, from Theorem 7.1, our instantiation (with and without the session resumption scheme for low-resource ED) is a 3-AKE-secure protocol according to Definition 5.4.

8 Conclusion

We have presented new results in three fields: firstly in cryptanalysis of security protocols, secondly in security models, and finally in key establishment mechanisms in the symmetric-key setting between two and three parties. In this conclusion, we summarise these results and open further perspectives.

8.1 Summary of the Results

In this thesis we have addressed the issue of establishing a secure tunnel for constrained devices. That is, devices with low resources with respect to computation, communication, and energy in particular.

At the beginning of our approach, we have observed that most of the current protocols either proposed or deployed are based on symmetric-key functions, and lack in providing strong security properties such as forward secrecy. Other protocols make use of asymmetric schemes which render them unsuitable to be implemented on very constrained devices. Overall, most of the current protocols for constrained devices trade security for efficiency (Chapter 1). We have (concretely) illustrated this assessment by the analysis of two widely deployed security protocols. Then, we have collected and devised otherwise the methodological tools necessary to assess the security of the constructions which were our goal in the end. Finally, we have proposed several key establishment protocols that feature enhanced properties (in terms of efficiency and security) with respect to existing ones.

Cryptanalysis of protocols. We have analysed two such protocols: LoRaWAN (Chapter 3) and SCP02 (Chapter 4). They are both widely deployed in LPWAN networks (LoRaWAN), and by the smart card industry (SCP02). The weaknesses we have highlighted lead to likely practical attacks against either protocol.

We have described how to break data integrity, data confidentiality, and the network availability against LoRaWAN 1.0. The aforementioned attacks, due to the protocol flaws, do not lean on potential implementation or hardware bugs, and are likely to be successful against any equipment implementing LoRaWAN 1.0. The success of the attacks is independent from the means used to protect the secret parameters (e.g., using a tamper resistant module such as a Secure Element). Furthermore, in one of the attack scenario, the attacker needs only to act on the air interface (to eavesdrop on data and interact with her target), but she does not need to get a physical access to any equipment (in particular the constrained device).

We have also presented how to apply a padding oracle attack against SCP02. This attack allows to decrypt, without the corresponding key, data transmitted in the secure tunnel. To illustrate the practical exploitability of this flaw, we have successfully attacked, in an experimental setting, 10 different models of smart card produced by six card manufacturers. To the best of our knowledge, this is the first successful attack against SCP02. Given that billions smart

cards are produced every year, the number of affected items, although difficult to estimate, is potentially high.

We have proposed practical recommendations aiming at thwarting, when possible, the attacks, and reported our findings to each consortium in charge of the development and the promotion of LoRaWAN and SCP02 respectively. LoRa Alliance has published a document aiming at strengthening the current version 1.0, and changes have been incorporated in the specification of version 1.1. Regarding SCP02, our results have contributed in the decision by GlobalPlatform to deprecate the protocol.

Security models. Considering the several vulnerabilities that impair these two real-life protocols, and in view of proposing efficient and secure key establishment protocols, we have devised two security models (Chapter 5). They capture the security properties that protocols must guarantee according to us, and incorporate the interleaved operations between the diverse components of an IoT network. We have applied one of these to a slightly modified version of LoRaWAN 1.1, and proved that, with a suitable choice of parameters and deployment, this modified version is secure in our model.

Forward secret symmetric-key protocols. Finally, we have presented two key establishment protocols for constrained devices. This first one, called SAKE, is solely built on symmetric-key functions (Chapter 6). Based on a shrewd synchronisation mechanism and a key evolving scheme, it guarantees forward secrecy.

3-AKE is a three-party protocol dedicated to IoT (Chapter 7). Also based on symmetric-key functions (regarding the computations done between the end-device and the back-end network), it guarantees forward secrecy, in contrast to widely deployed symmetric-key based IoT protocols. It also enables session resumption without impairing security (in particular, forward secrecy is maintained). This allows saving communication and computation cost which is advantageous for low-resource devices. This 3-party key exchange protocol can be applied in a real-case IoT deployment (i.e., involving numerous end-devices and servers) such that the latter inherits from the security properties of the protocol.

8.2 Perspectives and Open Problems

Further investigations. The SAKE (resp. SAKE-AM) protocol described in Chapter 6 requires five (resp. four) rounds, which can be reduced to four (resp. three) rounds if the two parties are synchronised with respect to their master keys evolution when a session starts. Each message of the protocol fulfills a specific task: party authentication, detecting desynchronisation, and then catching up. This eventually results in the forward secrecy property being ensured. Removing one message yields an attack, as shown by any of the numerous alternative versions we have analysed. Therefore, we do think that the figure of five rounds is the least achievable. Yet we do not formally prove it. This question of optimality deserves further investigation to conclude regarding the minimum number of rounds.

The 3-AKE security model presented in Chapter 5 (in order to analyse the three-party key establishment protocol described in Chapter 7) is based on the paradigm of indistinguishability of the session keys. In a protocol run, the computation of the "final" session key implies the involvement of "intermediary" session keys in several operations. This contradicts the possibility to prove the security of the protocol based on indistinguishability of the keys. In order to overcome this issue, we chose to limit the moments when the adversary can test a session key. A more suitable model remains to be devised (perhaps based on – variants of – the

models of Brzuska, Fischlin, Warinschi and Williams [BFWW11], Brzuska et al. [BFS+13], or Krawczyk [Kra16]), such that the security of the protocol be still defined on indistinguishability of keys, and the adversary be allowed to issue a Test-query at any moment.

Finally, implementing the SAKE and 3-AKE protocols in constrained devices (e.g., a passive RFID tag for SAKE) would be a beneficial proof of concept to concretely show the efficiency of both protocols.

Forward secrecy in the symmetric-key setting. Finding a symmetric equivalent of the Diffe-Hellman scheme would advantageously replace the latter. Some steps on this path have been done through algebraic generalisations of the DH scheme [Par15; PN18] or with the notion of "conversion function" (built from a symmetric-key primitive, such a function translates the ciphertext resulting from encryption under a specific key to the ciphertext corresponding to encryption with another key) [CK05]. So far, this line of work lead to mixed results, and remains to be continued.

Post-quantum cryptography. One of our goals was to overcome the challenge of devising key establishment protocols at the same time efficient on constrained devices, and with stronger security properties than existing protocols. Consequently, we have described new variants solely built on symmetric-key functions. Now a post-quantum era opens up. If the most common classical asymmetric schemes are all broken, the symmetric functions seem less vulnerable against quantum computing so far. But threats emerge against some primitives [KLLN16a; CNS17; KLLN16b].

Symmetric-key primitives (e.g., encryption or MAC functions) which are secure in a classic adversarial model can be broken in a quantum model. On these primitives functions can be built with an inner structure (e.g., a hidden shift) that can be revealed through the puzzling magic of quantum mechanics. This, in turn, discloses a secret parameter (such as a secret key or tweak) of the attacked primitive. The quantum setting differs from the classic one in many ways, and there are questions regarding the relevance of the superposition-based quantum security [Gag17]. Yet, some results [BHN+19] succeed in overcoming this issue, and propose a trade-off where the adversary (which has also access to quantum computing) is only granted classical queries (i.e., not in a quantum superposition of states).

This line of work on symmetric-key primitives is ongoing and can be extended to protocols. Beyond targeting the functions that constitutes a protocol, one can investigate the feasibility for a quantum adversary to take advantage of the protocol's technical specifics. More generally, continuing to study the exact quantum security of classic symmetric functions, analysing (symmetric) protocols (including those described in this thesis) in a quantum setting, and consequently devising suitable security models deserve attention.

Survivability of a security protocol. What may look surprising (amongst other assessments) in several existing IoT protocols is the possibility to almost trivially break them. This is not due to the way they are deployed and used, but rather to their specificities. For instance, very short parameters (2 and 3-byte long) are used in LoRaWAN 1.0. The attacks we present in Chapter 3 leverage this peculiarity. The protocol designers justify this choice by the small number of key exchanges a LoRaWAN device is supposed to execute in its whole lifespan. Yet, it is possible to compel a device to deviate from this expected behaviour. We have seen how the LoRaWAN protocol resists (badly). This raises a more generic question with regard to the way a protocol is supposed to behave when it is not used in ideal or expected conditions.

What we deal with here is not an issue related to *resilience* which may be defined as the

ability of a system under attack or in the presence of faults to recover all its capabilities from a degraded state to its nominal state [CMH+07]. The notion we are considering is not to be confused either with how much a protocol may withstand and maintain its nominal functionalities in the presence of faults or when facing attacks, which is related to *performability* [Mey80; Mey92]. Rather, our issue is related to the notion of *survivability* [KSS03; SKH+02]. Intuitively one would expect that the essential of the security properties be preserved, which leads to the problem of defining what is essential in security. This depends certainly on the intended goals of the protocol. The question of a survivable security protocol deserves further investigation. Regarding an environment where devices become pervasive, with no easy way to update the algorithms they implement, and physically liable to attacks, this issue is of particular concern.

Bibliography

[3rda] 3rd Generation Partnership Project. *Technical Specifications 33*. URL: http://www.3gpp.org/DynaReport/33-series.htm (cit. on pp. 20, 25, 140).

[3rdb] 3rd Generation Partnership Project. *Technical Specifications 35*. URL: http://www.3gpp.org/DynaReport/35-series.htm (cit. on pp. 20, 25, 140).

[AB00] Michel Abdalla and Mihir Bellare. *Increasing the Lifetime of a Key: a Comparative Analysis of the Security of Re-keying Techniques*. In: *ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. LNCS. Springer, Heidelberg, Dec. 2000, pp. 546–559 (cit. on pp. 142, 143).

[ABD+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. *Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice*. In: *ACM CCS 2015*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 5–17 (cit. on p. 159).

[ABF+19] Ghada Arfaoui, Xavier Bultel, Pierre-Alain Fouque, Adina Nedelcu, and Cristina Onete. *The privacy of the TLS 1.3 protocol*. Cryptology ePrint Archive, Report 2019/749. 2019. URL: https://eprint.iacr.org/2019/749 (cit. on p. 25).

[Acc15] Accenture. *Winning with the Industrial Internet of Things – How to accelerate the journey to productivity and growth*. 2015. URL: https://www.accenture.com/t00010101T000000Z__w__/it-it/_acnmedia/PDF-5/Accenture-Industrial-Internet-of-Things-Positioning-Paper-Report-2015.pdf (cit. on p. 165).

[ACD19] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. *The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol*. In: *EUROCRYPT 2019, Part I*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11476. LNCS. Springer, Heidelberg, May 2019, pp. 129–158 (cit. on p. 142).

[ACF19] Gildas Avoine, Sébastien Canard, and Loïc Ferreira. *IoT-Friendly AKE: Forward Secrecy and Session Resumption Meet Symmetric-Key Cryptography*. In: *ESORICS 2019, Part II*. Ed. by Kazue Sako, Steve Schneider, and Peter Y. A. Ryan. Vol. 11736. LNCS. Springer, Heidelberg, Sept. 2019, pp. 463–483 (cit. on pp. vii, viii, 27, 28, 99, 163).

[ACF20] G. Avoine, S. Canard, and L. Ferreira. *Symmetric-key Authenticated Key Exchange (SAKE) with Perfect Forward Secrecy*. In: *CT-RSA*. (To appear). 2020 (cit. on pp. viii, 28, 139).

[AF18a] Gildas Avoine and Loïc Ferreira. "Attacking GlobalPlatform SCP02-compliant Smart Cards Using a Padding Oracle Attack". In: *IACR TCHES* 2018.2 (2018). https://tches.iacr.org/index.php/TCHES/article/view/878, pp. 149–170. ISSN: 2569-2925 (cit. on pp. vi, 26, 27, 81).

[AF18b] Gildas Avoine and Loïc Ferreira. *Rescuing LoRaWAN 1.0*. In: *FC 2018*. Ed. by Sarah Meiklejohn and Kazue Sako. Vol. 10957. LNCS. Springer, Heidelberg, 2018, pp. 253–271 (cit. on pp. vi, 26, 27, 43, 73).

[AFM+16] Stéphanie Alt, Pierre-Alain Fouque, Gilles Macario-Rat, Cristina Onete, and Benjamin Richard. *A Cryptographic Analysis of UMTS/LTE AKA*. In: *ACNS 16*. Ed. by Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider. Vol. 9696. LNCS. Springer, Heidelberg, June 2016, pp. 18–35 (cit. on pp. 22, 23, 25, 100).

[AGJ19] Nimrod Aviram, Kai Gellert, and Tibor Jager. *Session Resumption Protocols and Efficient Forward Security for TLS 1.3 0-RTT*. In: *EUROCRYPT 2019, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. LNCS. Springer, Heidelberg, May 2019, pp. 117–150 (cit. on pp. 173, 175).

[AIES15] Gorka Irazoqui Apecechea, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. *Lucky 13 Strikes Back*. In: *ASIACCS 15*. Ed. by Feng Bao, Steven Miller, Jianying Zhou, and Gail-Joon Ahn. ACM Press, Apr. 2015, pp. 85–96 (cit. on p. 61).

[Ant17] Sebastian Anthony. *USB Killer now lets you fry most Lightning and USB-C devices for $55*. Ars Technica. 2017. URL: https://arstechnica.com/gadgets/2017/02/usb-killer-fry-lightning-usb-c-devices/ (cit. on p. 53).

[AP13] Nadhem J. AlFardan and Kenneth G. Paterson. *Lucky Thirteen: Breaking the TLS and DTLS Record Protocols*. In: *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013, pp. 526–540 (cit. on pp. 61, 96).

[AP16] Martin R. Albrecht and Kenneth G. Paterson. *Lucky Microseconds: A Timing Attack on Amazon's s2n Implementation of TLS*. In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 622–643 (cit. on p. 92).

[ASS09] Zahra Ahmadian, Somayeh Salimi, and Ahmad Salahi. *New attacks on UMTS network access*. In: *2009 Wireless Telecommunications Symposium*. 2009, pp. 1–6 (cit. on pp. 23, 25).

[Atm11] Atmel. *8-bit Atmel Microcontroller with 128KBytes In-System Programmable Flash*. 2011. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/doc2467.pdf (cit. on pp. 6, 7).

[Atm14] Atmel. *Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V*. 2014. URL: https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf (cit. on p. 6).

[AVTP+17] Ferran Adelantado, Xavier Vilajosana, Pere Tuset-Peiro, Borja Martinez, Joan Melià-Seguí, and Thomas Watteyne. "Understanding the Limits of LoRaWAN". In: *IEEE Communications Magazine* 55.9 (2017), pp. 34–40 (cit. on p. 76).

[AZM10] Aslan Askarov, Danfeng Zhang, and Andrew C. Myers. *Predictive black-box mitigation of timing channels*. In: *ACM CCS 2010*. Ed. by Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov. ACM Press, Oct. 2010, pp. 297–307 (cit. on p. 96).

[Ame09] American National Standards Institute. *ANSI X9.24-1:2009 Retail Financial Services Symmetric Key Management Part 1: Using Symmetric Techniques*. 2009 (cit. on p. 141).

[BBD+18] Karthikeyan Bhargavan, Ioana Boureanu, Antoine Delignat-Lavaud, Pierre-Alain Fouque, and Cristina Onete. *A Formal Treatment of Accountable Proxying Over TLS*. In: *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 799–816 (cit. on pp. 101, 112).

[BBF+17] Karthikeyan Bhargavan, Ioana Boureanu, Pierre-Alain Fouque, Cristina Onete, and Benjamin Richard. *Content delivery over TLS: a cryptographic analysis of keyless SSL*. In: *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2017, pp. 1–16 (cit. on pp. 100, 101, 106, 109, 111).

[BBK03] Elad Barkan, Eli Biham, and Nathan Keller. *Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication*. In: *CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. LNCS. Springer, Heidelberg, Aug. 2003, pp. 600–616 (cit. on p. 8).

[BCJ+06] Michael Backes, Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. *Cryptographically Sound Security Proofs for Basic and Public-Key Kerberos*. In: *ESORICS 2006*. Ed. by Dieter Gollmann, Jan Meier, and Andrei Sabelfeld. Vol. 4189. LNCS. Springer, Heidelberg, Sept. 2006, pp. 362–383 (cit. on p. 25).

[BCK98] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. *A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols (Extended Abstract)*. In: *30th ACM STOC*. ACM Press, May 1998, pp. 419–428 (cit. on p. 11).

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. *A Concrete Security Treatment of Symmetric Encryption*. In: *38th FOCS*. IEEE Computer Society Press, Oct. 1997, pp. 394–403 (cit. on pp. 32, 34, 50, 129, 136).

[BDL+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-speed high-security signatures". In: *Journal of Cryptographic Engineering* 2.2 (Sept. 2012), pp. 77–89 (cit. on p. 6).

[Bel98] Mihir Bellare. *Practice-Oriented Provable-Security (Invited Lecture)*. In: *ISW'97*. Ed. by Eiji Okamoto, George I. Davida, and Masahiro Mambo. Vol. 1396. LNCS. Springer, Heidelberg, Sept. 1998, pp. 221–231 (cit. on p. 50).

[Ber05] Daniel J. Bernstein. *The Poly1305-AES Message-Authentication Code*. In: *FSE 2005*. Ed. by Henri Gilbert and Helena Handschuh. Vol. 3557. LNCS. Springer, Heidelberg, Feb. 2005, pp. 32–49 (cit. on p. 7).

[Ber06] Daniel J. Bernstein. *Curve25519: New Diffie-Hellman Speed Records*. In: *PKC 2006*. Ed. by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. Vol. 3958. LNCS. Springer, Heidelberg, Apr. 2006, pp. 207–228 (cit. on p. 7).

[Ber08] Daniel J. Bernstein. *The Salsa20 Family of Stream Ciphers*. In: *New Stream Cipher Designs: The eSTREAM Finalists*. Ed. by Matthew Robshaw and Olivier Billet. Springer, 2008, pp. 84–97. URL: https://doi.org/10.1007/978-3-540-68351-3_8 (cit. on p. 6).

[Ber19] Daniel J. Bernstein. *CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness*. 2019. URL: https://competitions.cr.yp.to/caesar.html (cit. on p. 8).

[BFS+13] Christina Brzuska, Marc Fischlin, Nigel P. Smart, Bogdan Warinschi, and Stephen C. Williams. "Less is more: relaxed yet composable security notions for key exchange". In: *International Journal of Information Security* 12.4 (2013), pp. 267–297. URL: https://doi.org/10.1007/s10207-013-0192-y (cit. on pp. ix, 102, 197).

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. *Composability of Bellare-Rogaway key exchange protocols*. In: *ACM CCS 2011*. Ed. by Yan Chen, George Danezis, and Vitaly Shmatikov. ACM Press, Oct. 2011, pp. 51–62 (cit. on pp. ix, 197).

[BHN+19] Xavier Bonnetain, Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki, and André Schrottenloher. *Quantum Attacks without Superposition Queries: the Offline Simon Algorithm*. Cryptology ePrint Archive, Report 2019/614. 2019. URL: https://eprint.iacr.org/2019/614 (cit. on p. 197).

[Biz15] BiztechAfrica. *FastNet announces Africa's first dedicated M2M network and IoT developer academy*. BiztechAfrica. 2015. URL: http://www.biztechafrica.com/article/fastnet-announces-africas-first-dedicated-m2m-netw/10718/ (cit. on p. 45).

[BJS16] Christina Brzuska, Håkon Jacobsen, and Douglas Stebila. *Safely Exporting Keys from Secure Channels: On the Security of EAP-TLS and TLS Key Exporters*. In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 670–698 (cit. on pp. 36, 101, 185, 193).

[BJST08] Buno Blanchet, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. *Computationally sound mechanized proofs for basic and public-key Kerberos*. In: *ASIACCS 08*. Ed. by Masayuki Abe and Virgil Gligor. ACM Press, Mar. 2008, pp. 87–99 (cit. on p. 25).

[BK07] Alexandra Boldyreva and Virendra Kumar. *Provable-Security Analysis of Authenticated Encryption in Kerberos*. Cryptology ePrint Archive, Report 2007/234. http://eprint.iacr.org/2007/234. 2007 (cit. on p. 25).

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. *PRESENT: An Ultra-Lightweight Block Cipher*. In: *CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. LNCS. Springer, Heidelberg, Sept. 2007, pp. 450–466 (cit. on p. 7).

[BKN02] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. *Authenticated Encryption in SSH: Provably Fixing The SSH Binary Packet Protocol*. In: *ACM CCS 2002*. Ed. by Vijayalakshmi Atluri. ACM Press, Nov. 2002, pp. 1–11 (cit. on pp. 32, 100).

[BKN04] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. "Breaking and Provably Repairing the SSH Authenticated Encryption Scheme: A Case Study of the Encode-then-Encrypt-and-MAC Paradigm". In: *ACM Trans. Inf. Syst. Secur.* 7.2 (2004), pp. 206–241. URL: http://doi.acm.org/10.1145/996943.996945 (cit. on p. 96).

[Blo70] Burton H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors". In: *Communications of the ACM* 13.7 (1970), pp. 422–426. URL: http://doi.acm.org/10.1145/362686.362692 (cit. on pp. 63, 73).

[BM03a] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer, 2003 (cit. on p. 11).

[BM03b] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. ISC. Springer, Heidelberg, 2003. ISBN: 978-3-642-07716-6 (cit. on p. 140).

[BM99] Mihir Bellare and Sara K. Miner. *A Forward-Secure Digital Signature Scheme*. In: *CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Heidelberg, Aug. 1999, pp. 431–448 (cit. on p. 142).

[BMM+15] Christian Badertscher, Christian Matt, Ueli Maurer, Phillip Rogaway, and Björn Tackmann. *Augmented Secure Channels and the Goal of the TLS 1.3 Record Layer*. In: *ProvSec 2015*. Ed. by Man Ho Au and Atsuko Miyaji. Vol. 9451. LNCS. Springer, Heidelberg, Nov. 2015, pp. 85–104 (cit. on p. 193).

[BN00] Mihir Bellare and Chanathip Namprempre. *Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm*. In: *ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. LNCS. Springer, Heidelberg, Dec. 2000, pp. 531–545 (cit. on p. 136).

[BN08] Mihir Bellare and Chanathip Namprempre. "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm". In: *Journal of Cryptology* 21.4 (Oct. 2008), pp. 469–491 (cit. on pp. 32, 84, 96).

[BP10] Eric Brier and Thomas Peyrin. *A Forward-Secure Symmetric-Key Derivation Protocol - How to Improve Classical DUKPT*. In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Heidelberg, Dec. 2010, pp. 250–267 (cit. on pp. 141, 143).

[BP17] Alex Biryukov and Leo Perrin. *State of the Art in Lightweight Symmetric Cryptography*. Cryptology ePrint Archive, Report 2017/511. http://eprint.iacr.org/2017/511. 2017 (cit. on p. 8).

[BPG18] Ismail Butun, Nuno Pereira, and Mikael Gidlund. "Security Risk Analysis of LoRaWAN and Future Directions". In: *Future Internet* 11.1 (2018). URL: http://www.mdpi.com/1999-5903/11/1/3 (cit. on p. 75).

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. *Authenticated Key Exchange Secure against Dictionary Attacks*. In: *EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. LNCS. Springer, Heidelberg, May 2000, pp. 139–155 (cit. on p. 100).

[BR04] Mihir Bellare and Phillip Rogaway. *Code-Based Game-Playing Proofs and the Security of Triple Encryption*. Cryptology ePrint Archive, Report 2004/331. http://eprint.iacr.org/2004/331. 2004 (cit. on pp. 119, 153, 181).

[BR94] Mihir Bellare and Phillip Rogaway. *Entity Authentication and Key Distribution*. In: *CRYPTO'93*. Ed. by Douglas R. Stinson. Vol. 773. LNCS. Springer, Heidelberg, Aug. 1994, pp. 232–249 (cit. on pp. 11, 12, 25, 32, 100, 140).

[Bri16] Ken Briodagh. *Japan Opens New LoRaWAN Network for IoT Testing*. IoT Evolution World. 2016. URL: http://www.iotevolutionworld.com/iot/articles/423324-japan-opens-new-lorawan-network-iot-testing.htm (cit. on p. 45).

[BRS+15] Benjamin Buhrow, Paul Riemer, Mike Shea, Barry K. Gilbert, and Erik S. Daniel. *Block Cipher Speed and Energy Efficiency Records on the MSP430: System Design Trade-Offs for 16-Bit Embedded Applications*. In: *LATINCRYPT 2014*. Ed. by Diego F. Aranha and Alfred Menezes. Vol. 8895. LNCS. Springer, Heidelberg, Sept. 2015, pp. 104–123 (cit. on p. 6).

[BSNRN14] Alessandro Bruni, Michal Sojka, Flemming Nielson, and Hanne Riis Nielson. *Formal Security Analysis of the MaCAN Protocol*. In: *Integrated Formal Methods*. Ed. by Elvira Albert and Emil Sekerinski. Springer International Publishing, 2014, pp. 241–255 (cit. on p. 20).

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. *The SIMON and SPECK Families of Lightweight Block Ciphers*. Cryptology ePrint Archive, Report 2013/404. `http://eprint.iacr.org/2013/404`. 2013 (cit. on pp. 6, 7).

[BSS+15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. *SIMON and SPECK: Block Ciphers for the Internet of Things*. Cryptology ePrint Archive, Report 2015/585. `http://eprint.iacr.org/2015/585`. 2015 (cit. on p. 7).

[BSW01] Alex Biryukov, Adi Shamir, and David Wagner. *Real Time Cryptanalysis of A5/1 on a PC*. In: *FSE 2000*. Ed. by Bruce Schneier. Vol. 1978. LNCS. Springer, Heidelberg, Apr. 2001, pp. 1–18 (cit. on p. 8).

[BU02] John Black and Hector Urtubia. *Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption*. In: *USENIX Security 2002*. Ed. by Dan Boneh. USENIX Association, Aug. 2002, pp. 327–338 (cit. on pp. 85, 95).

[BWJM97] Simon Blake-Wilson, Don Johnson, and Alfred Menezes. *Key Agreement Protocols and Their Security Analysis*. In: *6th IMA International Conference on Cryptography and Coding*. Ed. by Michael Darnell. Vol. 1355. LNCS. Springer, Heidelberg, Dec. 1997, pp. 30–45 (cit. on pp. 11, 38, 100, 117, 159).

[BY03] Mihir Bellare and Bennet S. Yee. *Forward-Security in Private-Key Cryptography*. In: *CT-RSA 2003*. Ed. by Marc Joye. Vol. 2612. LNCS. Springer, Heidelberg, Apr. 2003, pp. 1–18 (cit. on pp. 141–143).

[CBH05] Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. *Examining Indistinguishability-Based Proof Models for Key Establishment Protocols*. In: *ASIACRYPT 2005*. Ed. by Bimal K. Roy. Vol. 3788. LNCS. Springer, Heidelberg, Dec. 2005, pp. 585–604 (cit. on p. 100).

[CC04] Zhaohui Cheng and Richard Comley. *Attacks On An ISO/IEC 11770-2 Key Establishment Protocol*. Cryptology ePrint Archive, Report 2004/249. `http://eprint.iacr.org/2004/249`. 2004 (cit. on p. 25).

[CF12] Cas Cremers and Michèle Feltz. *Beyond eCK: Perfect Forward Secrecy under Actor Compromise and Ephemeral-Key Reveal*. Cryptology ePrint Archive, Report 2012/416. `http://eprint.iacr.org/2012/416`. 2012 (cit. on p. 11).

[CF19] Sébastien Canard and Loïc Ferreira. *Extended 3-Party ACCE and Application to LoRaWAN 1.1*. In: *AFRICACRYPT 19*. Ed. by Johannes Buchmann, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 11627. LNCS. Springer, Heidelberg, July 2019, pp. 21–38 (cit. on pp. vi, vii, 26, 27, 43, 73, 99).

[CFPR15] Carlos Cid, Loïc Ferreira, Gordon Procter, and Matt J. B. Robshaw. *Algebraic Cryptanalysis and RFID Authentication*. In: *Radio Frequency Identification*. Ed. by Stefan Mangard and Patrick Schaumont. Springer International Publishing, 2015, pp. 104–121 (cit. on p. 9).

[CFR13] Sébastien Canard, Loïc Ferreira, and Matt Robshaw. *Improved (and Practical) Public-Key Authentication for UHF RFID Tags*. In: *Smart Card Research and Advanced Applications*. Ed. by Stefan Mangard. Springer, 2013, pp. 46–61 (cit. on p. 8).

[CGCD+17] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. *A Formal Security Analysis of the Signal Messaging Protocol*. In: *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2017, pp. 451–466 (cit. on p. 142).

[CH14] Cas Cremers and Marko Horvat. *Improving the ISO/IEC 11770 Standard for Key Management Techniques*. In: *Security Standardisation Research*. Ed. by Liqun Chen and Chris Mitchell. Springer International Publishing, 2014, pp. 215–235 (cit. on p. 25).

[CHH+17] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. *A Comprehensive Symbolic Analysis of TLS 1.3*. In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, 2017, pp. 1773–1788 (cit. on p. 25).

[CHVV03] Brice Canvel, Alain P. Hiltgen, Serge Vaudenay, and Martin Vuagnoux. *Password Interception in a SSL/TLS Channel*. In: *CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. LNCS. Springer, Heidelberg, Aug. 2003, pp. 583–599 (cit. on pp. 61, 88, 89, 92, 96).

[CK01] Ran Canetti and Hugo Krawczyk. *Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels*. In: *EUROCRYPT 2001*. Ed. by Birgit Pfitzmann. Vol. 2045. LNCS. Springer, Heidelberg, May 2001, pp. 453–474 (cit. on pp. 11, 100).

[CK02] Ran Canetti and Hugo Krawczyk. *Security Analysis of IKE's Signature-based Key-Exchange Protocol*. In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. http://eprint.iacr.org/2002/120/. Springer, Heidelberg, Aug. 2002, pp. 143–161 (cit. on p. 100).

[CK05] Debra L. Cook and Angelos D. Keromytis. *Conversion and Proxy Functions for Symmetric Key Ciphers*. In: *International Symposium on Information Technology: Coding and Computing – ITCC 2005*. Vol. 1. 2005, pp. 662–667. URL: https://doi.org/10.1109/ITCC.2005.115 (cit. on pp. x, 197).

[CLN16] Craig Costello, Patrick Longa, and Michael Naehrig. *Efficient Algorithms for Supersingular Isogeny Diffie-Hellman*. In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 572–601 (cit. on p. 160).

[CM08] Liqun Chen and Chris J. Mitchell. *Parsing ambiguities in authentication and key establishment protocols*. Cryptology ePrint Archive, Report 2008/419. http://eprint.iacr.org/2008/419. 2008 (cit. on p. 25).

[CMH+07] Piotr Cholda, Anders Mykkeltveit, Bjarne E. Helvik, Otto J. Wittner, and Andrzej Jajszczyk. "A Survey of Resilience Differentiation Frameworks in Communication Networks". In: *Commun. Surveys Tuts.* 9.4 (2007), pp. 32–55. URL: http://dx.doi.org/10.1109/COMST.2007.4444749 (cit. on pp. x, 198).

[CNS17] André Chailloux, María Naya-Plasencia, and André Schrottenloher. *An Efficient Quantum Collision Search Algorithm and Implications on Symmetric Cryptography*. In: *ASIACRYPT 2017, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. LNCS. Springer, Heidelberg, Dec. 2017, pp. 211–240 (cit. on pp. x, 8, 197).

[Cre09] Cas J. F. Cremers. *Session-state Reveal Is Stronger Than Ephemeral Key Reveal: Attacking the NAXOS Authenticated Key Exchange Protocol*. In: *ACNS 09*. Ed. by Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud. Vol. 5536. LNCS. Springer, Heidelberg, June 2009, pp. 20–33 (cit. on p. 100).

[Cre11] Cas J. F. Cremers. *Key Exchange in IPsec Revisited: Formal Analysis of IKEv1 and IKEv2*. In: *ESORICS 2011*. Ed. by Vijay Atluri and Claudia Díaz. Vol. 6879. LNCS. Springer, Heidelberg, 2011, pp. 315–334 (cit. on p. 100).

[CS15] Iwen Coisel and Ignacio Sanchez. *Improved Cryptanalysis of the DECT Standard Cipher*. In: *CHES 2015*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. LNCS. Springer, Heidelberg, Sept. 2015, pp. 269–286 (cit. on p. 9).

[DCK+15] Daniel Dinu, Yann Le Corre, Dmitry Khovratovich, Léo Perrin, Johann Großschädl, and Alex Biryukov. *Triathlon of Lightweight Block Ciphers for the Internet of Things*. Cryptology ePrint Archive, Report 2015/209. `http://eprint.iacr.org/2015/209`. 2015 (cit. on pp. 6, 7).

[DDSW11] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Marcel Winandy. *Privilege Escalation Attacks on Android*. In: *ISC 2010*. Ed. by Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic. Vol. 6531. LNCS. Springer, Heidelberg, Oct. 2011, pp. 346–360 (cit. on p. 92).

[DFGS15] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. *A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates*. In: *ACM CCS 2015*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 1197–1210 (cit. on p. 119).

[DFGS16] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. *A Cryptographic Analysis of the TLS 1.3 draft-10 Full and Pre-shared Key Handshake Protocol*. Cryptology ePrint Archive, Report 2016/081. `http://eprint.iacr.org/2016/081`. 2016 (cit. on p. 193).

[DFK+17] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella-Béguelin, Karthikeyan Bhargavan, Jianyang Pan, and Jean Karim Zinzindohoue. *Implementing and Proving the TLS 1.3 Record Layer*. In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 463–482 (cit. on p. 25).

[DH76] Whitfield Diffie and Martin E. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654 (cit. on pp. iv, 4, 140, 175).

[DH79] Whitfield Diffie and Martin E. Hellman. "Privacy and Authentication: An Introduction to Cryptography". In: *Proceedings of the IEEE* 67.3 (Mar. 1979), pp. 397–427 (cit. on pp. 46, 50).

[DJ14] Mohammad Sadeq Dousti and Rasool Jalili. *FORSAKES: A Forward-Secure Authenticated Key Exchange Protocol Based on Symmetric Key-Evolving Schemes*. Cryptology ePrint Archive, Report 2014/123. http://eprint.iacr.org/2014/123. 2014 (cit. on pp. 12, 22, 25, 140, 143).

[DM04] Peter C. Dillinger and Panagiotis Manolios. *Bloom Filters in Probabilistic Verification*. In: *Formal Methods in Computer-Aided Design*. Vol. 3312. Springer, 2004, pp. 367–381 (cit. on pp. 63, 73).

[DN18a] Tahsin C. M. Dönmez and Ethiopia Nigussie. *Security of Join Procedure and its Delegation in LoRaWAN v1.1*. In: *FNC/MobiSPC*. Vol. 134. 2018, pp. 204–211. URL: https://doi.org/10.1016/j.procs.2018.07.202 (cit. on p. 76).

[DN18b] Tahsin C. M. Dönmez and Ethiopia Nigussie. *Security of LoRaWAN v1.1 in Backward Compatibility Scenarios*. In: *FNC/MobiSPC*. Ed. by Elhadi Shakshuki and Ansar Yasar. Vol. 134. 2018, pp. 51–58. URL: https://doi.org/10.1016/j.procs.2018.07.143 (cit. on p. 76).

[DPW11] Jean Paul Degabriele, Kenneth G. Paterson, and Gaven J. Watson. "Provable Security in the Real World". In: *IEEE Security and Privacy* 9.3 (May 2011), pp. 33–41 (cit. on p. 50).

[DR08] Tim Dierks and Eric Rescorla. *The Transport Layer Security (TLS) Protocol – Version 1.2*. 2008. URL: https://tools.ietf.org/html/rfc5246 (cit. on pp. 4, 11, 13, 38, 82, 84, 100, 107, 111, 119, 175).

[DvW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. "Authentication and Authenticated Key Exchanges". In: *Designs, Codes and Cryptography* 2.2 (June 1992), pp. 107–125 (cit. on pp. iv, 9, 140).

[Dwo01] Morris Dworkin. *NIST Special Publication 800-38A Recommendation for Block Cipher Modes of Operation – Methods and Techniques*. Dec. 2001. URL: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf (cit. on pp. 46, 50).

[Dwo05] Morris Dworkin. *NIST Special Publication 800-38B Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*. May 2005. URL: http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf (cit. on p. 46).

[DY81] Danny Dolev and Andrew Chi-Chih Yao. *On the Security of Public Key Protocols (Extended Abstract)*. In: *22nd FOCS*. IEEE Computer Society Press, Oct. 1981, pp. 350–357 (cit. on p. 22).

[Dön19] Tahsin Dönmez. Personal communication. 2019 (cit. on p. 77).

[EBPG18] Mohamed Eldefrawy, Ismail Butun, Nuno Pereira, and Mikael Gidlund. "Formal Security Analysis of LoRaWAN". In: *Computer Networks* 148 (2018), pp. 328–339. URL: https://doi.org/10.1016/j.comnet.2018.11.017 (cit. on p. 78).

[EKP+07] Thomas Eisenbarth, Sandeep Kumar, Christof Paar, Axel Poschmann, and Leif Uhsadel. "A Survey of Lightweight-Cryptography Implementations". In: *IEEE Des. Test* 24.6 (2007), pp. 522–533. URL: https://doi.org/10.1109/MDT.2007.178 (cit. on p. 7).

[ET05] Pasi Eronen and Hannes Tschofenig. *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*. RFC 4279. 2005 (cit. on pp. 13, 25, 112).

[ECR08] ECRYPT network. *The ECRYPT Stream Cipher Project*. 2008. URL: https://www.ecrypt.eu.org/stream/ (cit. on p. 8).

[ETS17] ETSI. *Smart Cards; Secure packet structure for UICC based applications (release 12)*. TS 102.225, v12.1.0 (2014-10). 2017 (cit. on p. 82).

[Fea16] Nicholas Fearn. *Orange deploys LoRa network in thousands of French towns*. Internet of Business. 2016. URL: https://internetofbusiness.com/orange-lora-network-france/ (cit. on p. 45).

[Fei73] Horst Feistel. "TCryptography and Data Security". In: *Scientific American* 228.5 (1973), pp. 15–23 (cit. on p. 4).

[FOR16] Pierre-Alain Fouque, Cristina Onete, and Benjamin Richard. *Achieving Better Privacy for the 3GPP AKA Protocol*. Cryptology ePrint Archive, Report 2016/480. http://eprint.iacr.org/2016/480. 2016 (cit. on p. 100).

[FS99] Niels Ferguson and Bruce Schneier. *A Cryptographic Evaluation of IPsec*. 1999 (cit. on p. 100).

[GAB19] Utku Gulen, Abdelrahman Alkhodary, and Selcuk Baktir. "Implementing RSA for Wireless Sensor Nodes". In: *Sensors* 19.13 (2019). URL: https://doi.org/10.3390/s19132864 (cit. on p. 6).

[Gag17] Tommaso Gagliardoni. *Quantum Security of Cryptographic Primitives*. PhD thesis. Technische Universität Darmstadt, 2017. URL: http://tuprints.ulb.tu-darmstadt.de/6019/ (cit. on p. 197).

[GL04] Marc Girault and David Lefranc. *Public Key Authentication with One (Online) Single Addition*. In: *CHES 2004*. Ed. by Marc Joye and Jean-Jacques Quisquater. Vol. 3156. LNCS. Springer, Heidelberg, Aug. 2004, pp. 413–427 (cit. on p. 8).

[Glo14] GlobalPlatform. *GlobalPlatform Technology – Secure Channel Protocol '03' – Card Specification v2.2 – Amendment D*. version 1.1.1, GPC_SPE_014. 2014. URL: http://www.globalplatform.org/specificationscard.asp (cit. on pp. 14, 25, 82).

[Glo18] GlobalPlatform. *GlobalPlatform Technology – Card Specification – Version 2.3.1*. Reference GPC_SPE_034. 2018. URL: https://www.globalplatform.org/specificationscard.asp (cit. on pp. 14, 25, 82, 140).

[GMVV12] Bogdan Groza, Pal-Stefan Murvay, Anthony Van Herrewege, and Ingrid Verbauwhede. *LiBrA-CAN: A Lightweight Broadcast Authentication Protocol for Controller Area Networks*. In: *CANS 12*. Ed. by Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis. Vol. 7712. LNCS. Springer, Heidelberg, Dec. 2012, pp. 185–200 (cit. on p. 20).

[Got] *Compact server for private LoRaWAN networks*. URL: https://github.com/gotthardp/lorawan-server (cit. on p. 55).

[GPS06] Marc Girault, Guillaume Poupard, and Jacques Stern. "On the Fly Authentication and Signature Schemes Based on Groups of Unknown Order". In: *Journal of Cryptology* 19.4 (Oct. 2006), pp. 463–487 (cit. on p. 8).

[Gro96] Lov K. Grover. *A Fast Quantum Mechanical Algorithm for Database Search*. In: *28th ACM STOC*. ACM Press, May 1996, pp. 212–219 (cit. on p. 160).

[GSM16] GSMA. *3GPP Low Power Wide Area Technologies – GSMA White Paper*. 2016 (cit. on p. 22).

[GSM19] GSMA. *The Mobile Economy 2019*. 2019. URL: https://www.gsma.com/r/mobileeconomy/ (cit. on pp. 82, 95).

[Gün90] Christoph G. Günther. *An Identity-Based Key-Exchange Protocol*. In: *EUROCRYPT'89*. Ed. by Jean-Jacques Quisquater and Joos Vandewalle. Vol. 434. LNCS. Springer, Heidelberg, Apr. 1990, pp. 29–37 (cit. on pp. iv, 9, 140).

[Har14] Thomas Hardjono. *Kerberos for Internet-of-Things*. 2014. URL: https://kit.mit.edu/sites/default/files/documents/Kerberos_Internet_of%20Things.pdf (cit. on p. 13).

[HC14] Chan-Kyu Han and Hyoung-Kee Choi. "Security Analysis of Handover Key Management in 4G LTE/SAE Networks". In: *IEEE Transactions on Mobile Computing* 13.2 (2014), pp. 457–468 (cit. on pp. 23, 25).

[HFPM18] George Hatzivasilis, Konstantinos Fysarakis, Ioannis Papaefstathiou, and Charalampos Manifavas. "A review of lightweight block ciphers". In: *Journal of Cryptographic Engineering* 8.2 (June 2018), pp. 141–184 (cit. on p. 8).

[HGFS15] Clemens Hlauschek, Markus Gruber, Florian Fankhauser, and Christian Schanes. *Prying Open Pandora's Box: KCI Attacks Against TLS*. In: *Proceedings of the 9th USENIX Conference on Offensive Technologies*. WOOT'15. USENIX Association, 2015 (cit. on p. 159).

[HHR+08] Daniel Halperin, Thomas S. Heydt-Benjamin, Benjamin Ransford, Shane S. Clark, Benessa Defend, Will Morgan, Kevin Fu, Tadayoshi Kohno, and William H. Maisel. *Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses*. In: *2008 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2008, pp. 129–142 (cit. on p. 5).

[HS13] Michael Hutter and Peter Schwabe. *NaCl on 8-Bit AVR Microcontrollers*. In: *AFRICACRYPT 13*. Ed. by Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien. Vol. 7918. LNCS. Springer, Heidelberg, June 2013, pp. 156–172 (cit. on pp. 6, 7).

[HSD+05] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell. *A Modular Correctness Proof of IEEE 802.11i and TLS*. In: *ACM CCS 2005*. Ed. by Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels. ACM Press, Nov. 2005, pp. 2–15 (cit. on p. 25).

[Hun17] Derek Hunt. *Certification Deep Dive*. LoRa Alliance. 2017. URL: https://lora-alliance.org/resource-hub/certification-deep-dive (cit. on p. 54).

[HW08] Georg Hofferek and Johannes Wolkerstorfer. *Coupon Recalculation for the GPS Authentication Scheme*. In: *Smart Card Research and Advanced Applications*. Ed. by Gilles Grimaud and François-Xavier Standaert. Vol. 5189. LNCS. Springer, 2008, pp. 162–175 (cit. on p. 8).

[HW18] Jialuo Han and Jidong Wang. *An Enhanced Key Management Scheme for LoRaWAN*. In: *Security, Privacy, and Anonymity in Computation, Communication, and Storage*. Ed. by Guojun Wang, Jinjun Chen, and Laurence T. Yang. Springer International Publishing, 2018, pp. 407–416 (cit. on p. 78).

[IK03a] Tetsu Iwata and Kaoru Kurosawa. *OMAC: One-Key CBC MAC*. In: *FSE 2003*. Ed. by Thomas Johansson. Vol. 2887. LNCS. Springer, Heidelberg, Feb. 2003, pp. 129–153 (cit. on p. 50).

[IK03b] Tetsu Iwata and Kaoru Kurosawa. *OMAC: One-Key CBC MAC – Addendum*. 2003. URL: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/omac/omac-ad.pdf (cit. on p. 50).

[IK03c] Tetsu Iwata and Kaoru Kurosawa. *Stronger Security Bounds for OMAC, TMAC, and XCBC*. In: *INDOCRYPT 2003*. Ed. by Thomas Johansson and Subhamoy Maitra. Vol. 2904. LNCS. Springer, Heidelberg, Dec. 2003, pp. 402–415 (cit. on pp. 50, 136).

[i-s18] i-scoop. *The Industrial Internet of Things (IIoT): the business guide to Industrial IoT*. 2018. URL: https://www.i-scoop.eu/internet-of-things-guide/industrial-internet-things-iiot-saving-costs-innovation/ (cit. on p. 165).

[IEE04] IEEE. *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements*. 2004 (cit. on pp. 20, 23, 25).

[IHS17] IHS Markit. *The Internet of Things: a movement, not a market*. 2017. URL: https://cdn.ihs.com/www/pdf/IoT_ebook.pdf (cit. on p. 4).

[Int08] International Organization for Standardization. *ISO/IEC 11770-2 – Information technology – Security techniques – Key Management – Part 2: Mechanisms using Symmetric Techniques*. 2008 (cit. on pp. 12, 25, 140).

[Int11] International Organization for Standardization. *ISO/IEC 9797-1:2011 – Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher*. 2011 (cit. on p. 83).

[Int17] International Organization for Standardization. *ISO/IEC 10116:2017 – Information technology – Security techniques – Modes of operation for an n-bit block cipher*. 2017 (cit. on pp. 83, 95).

[IoT19] IoT.Business.News. *MWC 2019 – LoRa Alliance: interview with CEO, Donna Moore*. 2019. URL: https://iotbusinessnews.com/2019/03/13/37011-mwc-2019-lora-alliance-interview-with-ceo-donna-moore/ (cit. on p. 45).

[JD11] David Jao and Luca De Feo. *Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies*. In: *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*. Ed. by Bo-Yin Yang. Springer, Heidelberg, 2011, pp. 19–34 (cit. on p. 160).

[JKSS11] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. *On the Security of TLS-DHE in the Standard Model*. Cryptology ePrint Archive, Report 2011/219. http://eprint.iacr.org/2011/219. 2011 (cit. on pp. 11, 28, 32, 36, 38, 100, 106, 111, 117, 119).

[Jon03] Jakob Jonsson. *On the Security of CTR + CBC-MAC*. In: *SAC 2002*. Ed. by Kaisa Nyberg and Howard M. Heys. Vol. 2595. LNCS. Springer, Heidelberg, Aug. 2003, pp. 76–93 (cit. on p. 84).

[KE10] Hugo Krawczyk and Pasi Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. RFC 5869. 2010. URL: https://tools.ietf.org/html/rfc5869 (cit. on p. 16).

[Ker83a] Auguste Kerckhoffs. "La cryptographie militaire". In: *Journal des sciences militaires* 9 (1883), pp. 5–38 (cit. on p. 4).

[Ker83b] Auguste Kerckhoffs. "La cryptographie militaire – Seconde partie". In: *Journal des sciences militaires* 9 (1883), pp. 161–191 (cit. on p. 4).

[KHN+14] Charlie Kaufman, Paul Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen. *Internet Key Exchange Protocol Version 2 (IKEv2)*. RFC 7296. 2014. URL: https://tools.ietf.org/html/rfc7296 (cit. on pp. 4, 175).

[Kim16] Gary Kim. *Tata to deploy LoRa network for IoT*. Spectrum Futures. 2016. URL: http://spectrumfutures.org/tata-to-deploy-lora-network-for-iot/ (cit. on p. 45).

[Kin16] Sean Kinney. *100 US cities covered by Senet LoRa network for IoT*. RCR Wireless News. 2016. URL: http://www.rcrwireless.com/20160615/internet-of-things/100-u-s-cities-covered-senet-lora-network-iot-tag17 (cit. on p. 45).

[Kiv] Anton Kivva. *The banker that can steal anything*. 20/09/2016. URL: https://securelist.com/the-banker-that-can-steal-anything/76101/ (cit. on p. 92).

[KJ17] Burton S. Kaliski Jr. *A Quantum "Magic Box" for the Discrete Logarithm Problem*. Cryptology ePrint Archive, Report 2017/745. 2017. URL: https://eprint.iacr.org/2017/745 (cit. on p. 160).

[KLLN16a] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. *Breaking Symmetric Cryptosystems Using Quantum Period Finding*. In: *CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. LNCS. Springer, Heidelberg, Aug. 2016, pp. 207–237 (cit. on pp. x, 8, 160, 197).

[KLLN16b] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. "Quantum Differential and Linear Cryptanalysis". In: *IACR Trans. Symm. Cryptol.* 2016.1 (2016). http://tosc.iacr.org/index.php/ToSC/article/view/536, pp. 71–94. ISSN: 2519-173X (cit. on pp. x, 8, 197).

[KMSS15] Dennis Kupser, Christian Mainka, Jörg Schwenk, and Juraj Somorovsky. *How to Break XML Encryption – Automatically*. In: *Proceedings of the 9th USENIX Conference on Offensive Technologies*. WOOT'15. USENIX Association, 2015, pp. 11–11. URL: http://dl.acm.org/citation.cfm?id=2831211.2831222 (cit. on p. 96).

[KOY03] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. *Forward Secrecy in Password-Only Key Exchange Protocols*. In: *SCN 02*. Ed. by Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano. Vol. 2576. LNCS. Springer, Heidelberg, Sept. 2003, pp. 29–44 (cit. on p. 11).

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. *On the Security of the TLS Protocol: A Systematic Analysis*. In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 429–448 (cit. on pp. 32, 100).

[Kra01] Hugo Krawczyk. *The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)* In: *CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. LNCS. Springer, Heidelberg, Aug. 2001, pp. 310–331 (cit. on p. 84).

[Kra03] Hugo Krawczyk. *SIGMA: The "SIGn-and-MAc" Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols*. In: *CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. LNCS. Springer, Heidelberg, Aug. 2003, pp. 400–425 (cit. on p. 112).

[Kra05] Hugo Krawczyk. *HMQV: A High-Performance Secure Diffie-Hellman Protocol*. Cryptology ePrint Archive, Report 2005/176. http://eprint.iacr.org/2005/176. 2005 (cit. on p. 11).

[Kra16] Hugo Krawczyk. *A Unilateral-to-Mutual Authentication Compiler for Key Exchange (with Applications to Client Authentication in TLS 1.3)*. In: *ACM CCS 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1438–1450 (cit. on pp. ix, 102, 197).

[Kru09] John Krumm. *Ubiquitous Computing Fundamentals*. 1st. Chapman & Hall/CRC, 2009 (cit. on p. 4).

[KS17] Jaehyu Kim and JooSeok Song. "A Dual Key-Based Activation Scheme for Secure LoRaWAN". In: *Wireless Communications and Mobile Computing* 2017 (2017). URL: https://doi.org/10.1155/2017/6590713 (cit. on p. 77).

[KSS03] John C. Knight, Elisabeth A. Strunk, and Kevin J. Sullivan. *Towards a rigorous definition of information system survivability*. In: *Proceedings of DARPA Information Survivability Conference and Exposition – DISCEX'03*. Vol. 1. IEEE, 2003, pp. 78–89. URL: https://doi.org/10.1109/DISCEX.2003.1194874 (cit. on pp. x, 198).

[KSS13] Florian Kohlar, Sven Schäge, and Jörg Schwenk. *On the Security of TLS-DH and TLS-RSA in the Standard Model*. Cryptology ePrint Archive, Report 2013/367. http://eprint.iacr.org/2013/367. 2013 (cit. on pp. 111, 119).

[LBM07] Tri Van Le, Mike Burmester, and Breno de Medeiros. *Universally composable and forward-secure RFID authentication and authenticated key exchange*. In: *ASIACCS 07*. Ed. by Feng Bao and Steven Miller. ACM Press, Mar. 2007, pp. 242–252 (cit. on pp. 14, 23, 25, 140, 143).

[Lif16] Renaud Lifchitz. *Security review of LoRaWAN networks*. Hardwear.io. 2016. URL: https://speakerdeck.com/rlifchitz/security-review-of-lorawan-networks (cit. on p. 74).

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. *Stronger Security of Authenticated Key Exchange*. In: *ProvSec 2007*. Ed. by Willy Susilo, Joseph K. Liu, and Yi Mu. Vol. 4784. LNCS. Springer, Heidelberg, Nov. 2007, pp. 1–16 (cit. on pp. 11, 100).

[LOL12] Conrado Porto Lopes Gouvêa, Leonardo B. Oliveira, and Julio Cesar López-Hernández. "Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller". In: *Journal of Cryptographic Engineering* 2.1 (May 2012), pp. 19–29 (cit. on p. 7).

[LoR] LoRa Alliance. *LoRaWAN Certified Products*. Last consulted December 8, 2017. URL: https://www.lora-alliance.org/certified-products (cit. on p. 54).

[LoR17] LoRa Alliance Technical committee. *LoRaWAN 1.0.2 Regional Parameters*. LoRa Alliance, version 1.0, revision B. 2017. URL: https://lora-alliance.org/resource-hub/lorawantm-regional-parameters-v102rb (cit. on p. 45).

[LoR18a] LoRa Alliance Technical committee. *LoRaWAN 1.0.3 Specification*. LoRa Alliance. 2018. URL: https://lora-alliance.org/resource-hub/lorawanr-specification-v103 (cit. on pp. 22, 24, 25, 45, 165).

[LoR18b] LoRa Alliance Technical committee. *Technical Recommendations for Preventing State Synchronization Issues around LoRaWAN 1.0.x Join Procedure*. LoRa Alliance, version 1.0.0. 2018. URL: https://lora-alliance.org/resource-hub/technical-recommendations-preventing-state-synchronization-issues-around-lorawantm-10x (cit. on pp. 27, 63, 70, 73).

[LSY+14] Yong Li, Sven Schäge, Zheng Yang, Florian Kohlar, and Jörg Schwenk. *On the Security of the Pre-shared Key Ciphersuites of TLS*. In: *PKC 2014*. Ed. by Hugo Krawczyk. Vol. 8383. LNCS. Springer, Heidelberg, Mar. 2014, pp. 669–684 (cit. on pp. 11, 25, 40).

[Lun17] Lucas Lundgren. *Taking over the world through MQTT – Aftermath*. Black Hat USA. 2017 (cit. on pp. 60, 70).

[LWG14] Zhe Liu, Erich Wenger, and Johann Großschädl. *MoTE-ECC: Energy-Scalable Elliptic Curve Cryptography for Wireless Sensor Networks*. In: *ACNS 14*. Ed. by Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay. Vol. 8479. LNCS. Springer, Heidelberg, June 2014, pp. 361–379 (cit. on pp. 6, 7).

[Mar16] Sue Marek. *SK Telecom & KPN Deploy Nationwide LoRa IoT Networks*. sdx central. 2016. URL: https://www.sdxcentral.com/articles/news/sk-telecom-kpn-deploy-nationwide-lorawan-iot-networks/2016/07/ (cit. on p. 45).

[McG08] David McGrew. *An Interface and Algorithms for Authenticated Encryption*. RFC 5116. 2008. URL: https://tools.ietf.org/html/rfc5116 (cit. on p. 119).

[MEM] MEMSIC. *TelosB – TelosB Mote Platform*. URL: http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf (cit. on p. 7).

[Mey80] John F. Meyer. "On Evaluating the Performability of Degradable Computing Systems". In: *IEEE Trans. Comput.* 29.8 (1980), pp. 720–731. URL: https://doi.org/10.1109/TC.1980.1675654 (cit. on pp. x, 198).

[Mey92] John F. Meyer. "Performability: a retrospective and some pointers to the future". In: *Performance Evaluation* 14.3–4 (1992), pp. 139–156. URL: https://doi.org/10.1016/0166-5316(92)90002-X (cit. on pp. x, 198).

[MGSP08] Giacomo de Meulenaer, François Gosset, François-Xavier Standaert, and Olivier Pereira. *On the Energy Cost of Communications and Cryptography in Wireless Sensor Networks*. In: *IEEE International Workshop on Security and Privacy in Wireless and Mobile Computing, Networking and Communications (SecPriWiMob'2008)*. 2008, pp. 580–585 (cit. on pp. 6, 7).

[Mil16a] Robert Miller. *LoRa Security – Building a Secure LoRa Solution*. Whitepaper, MWR Labs. Mar. 2016. URL: https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-LoRa-security-guide-1.2-2016-03-22.pdf (cit. on p. 74).

[Mil16b] Robert Miller. *LoRa the Explorer – Attacking and Defending LoRa Systems*. Information Security Conference – SyScan360. 2016. URL: https://www.syscan360.org/slides/2016_SG_Robert_Miller_LoRa_the_Explorer-Attacking_and_Defending_LoRa_systems.pdf (cit. on p. 74).

[MMDF+12] Andres Molina-Markham, George Danezis, Kevin Fu, Prashant J. Shenoy, and David E. Irwin. *Designing Privacy-Preserving Smart Meters with Low-Cost Microcontrollers*. In: *FC 2012*. Ed. by Angelos D. Keromytis. Vol. 7397. LNCS. Springer, Heidelberg, 2012, pp. 239–253 (cit. on p. 8).

[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. *Pushing the Limits: A Very Compact and a Threshold Implementation of AES*. In: *EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. LNCS. Springer, Heidelberg, May 2011, pp. 69–88 (cit. on p. 6).

[MS08] Anish Mathuria and G. Sriram. *New attacks on ISO key establishment protocols*. Cryptology ePrint Archive, Report 2008/336. `http://eprint.iacr.org/2008/336`. 2008 (cit. on p. 25).

[MSG+16] Eduard Marin, Dave Singelée, Flavio D. Garcia, Tom Chothia, Rik Willems, and Bart Preneel. *On the (in)Security of the Latest Generation Implantable Cardiac Defibrillators and How to Secure Them*. In: *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACSAC '16. ACM, 2016, pp. 226–236. URL: `http://doi.acm.org/10.1145/2991079.2991094` (cit. on p. 5).

[MSW08] Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. *A Modular Security Analysis of the TLS Handshake Protocol*. In: *ASIACRYPT 2008*. Ed. by Josef Pieprzyk. Vol. 5350. LNCS. Springer, Heidelberg, Dec. 2008, pp. 55–73 (cit. on p. 100).

[MSY+16] Eduard Marin, Dave Singelée, Bohan Yang, Ingrid Verbauwhede, and Bart Preneel. *On the Feasibility of Cryptography for a Wireless Insulin Pump System*. In: *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. CODASPY '16. ACM, 2016, pp. 113–120. URL: `http://doi.acm.org/10.1145/2857705.2857746` (cit. on p. 5).

[MV15] Charlie Miller and Chris Valasek. *Remote Exploitation of an Unaltered Passenger Vehicle*. 2015. URL: `http://illmatics.com/Remote%20Car%20Hacking.pdf` (cit. on p. 5).

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. 1st. CRC Press, Inc., 1996 (cit. on p. 11).

[MW04] Ulrike Meyer and Susanne Wetzel. *A Man-in-the-middle Attack on UMTS*. In: *Proceedings of the 3rd ACM Workshop on Wireless Security – WiSe '04*. Philadelphia, PA, USA: ACM, 2004, pp. 90–97 (cit. on pp. 23, 25).

[MWES06] Joshua Mason, Kathryn Watkins, Jason Eisner, and Adam Stubblefield. *A natural language approach to automated cryptanalysis of two-time pads*. In: *ACM CCS 2006*. Ed. by Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati. ACM Press, 2006, pp. 235–244 (cit. on pp. 54, 71).

[Nan09] Mridul Nandi. "Improved Security Analysis for OMAC as a Pseudo Random Function". In: *Journal of Mathematical Cryptology* 3.2 (Aug. 2009), pp. 133–148 (cit. on p. 50).

[Nat01] National Institute Of Standards and Technology. *NIST FIPS 197 Specification for the Advanced Encryption Standard (AES)*. 2001. URL: `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf` (cit. on pp. 7, 46).

[Nat04] National Institute Of Standards and Technology. *NIST Special Publication 800-38C Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*. May 2004. URL: http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf (cit. on p. 84).

[Nat19] National Institute Of Standards and Technology. *Lightweight Cryptography*. 2019. URL: https://csrc.nist.gov/Projects/Lightweight-Cryptography (cit. on p. 8).

[Nat99] National Institute Of Standards and Technology. *Data Encryption Standard (DES)*. 1999. URL: https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf (cit. on p. 4).

[NCOS16] Erick Nascimento, Lukasz Chmielewski, David Oswald, and Peter Schwabe. *Attacking Embedded ECC Implementations Through cmov Side Channels*. In: *SAC 2016*. Ed. by Roberto Avanzi and Howard M. Heys. Vol. 10532. LNCS. Springer, Heidelberg, Aug. 2016, pp. 99–119 (cit. on p. 7).

[NH09] Christoph Nagl and Michael Hutter. *Coupon Recalculation for the Schnorr and GPS Identification Scheme: A Performance Evaluation*. In: *Workshop on RFID Security 2009 – RFIDSec 2009*. Ed. by Lejla Batina. 2009, pp. 1–10 (cit. on p. 8).

[NL15] Yoav Nir and Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 7539. 2015. URL: https://tools.ietf.org/html/rfc7539 (cit. on p. 119).

[NLG+17] David Naylor, Richard Li, Christos Gkantsidis, Thomas Karagiannis, and Peter Steenkiste. *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*. CoNEXT '17. ACM, 2017, pp. 88–100. URL: http://doi.acm.org/10.1145/3143361.3143383 (cit. on p. 101).

[NR16] Stefan Nürnberger and Christian Rossow. - *vatiCAN - Vetted, Authenticated CAN Bus*. In: *CHES 2016*. Ed. by Benedikt Gierlichs and Axel Y. Poschmann. Vol. 9813. LNCS. Springer, Heidelberg, Aug. 2016, pp. 106–124 (cit. on p. 20).

[NSV+15] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. *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*. SIGCOMM '15. ACM, 2015, pp. 199–212. URL: http://doi.acm.org/10.1145/2785956.2787482 (cit. on pp. 100, 101).

[NYHR05] Clifford Neuman, Tom Yu, Sam Hartman, and Ken Raeburn. *The Kerberos Network Authentication Service (V5)*. RFC 4120. 2005. URL: https://tools.ietf.org/html/rfc4120 (cit. on pp. 7, 13, 25).

[OA16] Orange and Actility. *LoRa Device Developer Guide – Orange Connected Objects & Partnerships*. 2016. URL: https://developer.orange.com/wp-content/uploads/LoRa-Device-Developer-Guide-Orange.pdf (cit. on p. 45).

[Par15] Juha Partala. *Algebraic methods for cryptographic key exhange*. PhD thesis. University of Oulu, 2015. URL: http://urn.fi/urn:isbn:9789526207445 (cit. on pp. x, 197).

[PM16] Trevor Perrin and Moxie Marlinspike. *The Double Ratchet Algorithm*. Revision 1, 20/11/2016. 2016. URL: https://signal.org/docs/specifications/doubleratchet/ (cit. on p. 142).

[PN18] Jacques Patarin and Valérie Nachef. *Commutativity, Associativity, and Public Key Cryptography*. In: *Number-Theoretic Methods in Cryptology*. Ed. by Jerzy Kaczorowski, Josef Pieprzyk, and Jacek Pomykała. Springer International Publishing, 2018, pp. 104–117 (cit. on pp. x, 197).

[PS04] Taejoon Park and Kang G. Shin. "LiSP: A Lightweight Security Protocol for Wireless Sensor Networks". In: *ACM Trans. Embed. Comput. Syst.* 3.3 (2004), pp. 634–660 (cit. on pp. 19, 25, 140).

[PS18] Christopher Patton and Thomas Shrimpton. *Partially Specified Channels: The TLS 1.3 Record Layer without Elision*. In: *ACM CCS 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM Press, Oct. 2018, pp. 1415–1428 (cit. on p. 25).

[PST+02] Adrian Perrig, Robert Szewczyk, J.D. Tygar, Victor Wen, and David E. Culler. "SPINS: Security Protocols for Sensor Networks". In: *Wireless Networks* 8.5 (2002), pp. 521–534 (cit. on pp. 18, 25, 140).

[PW08] Kenneth G. Paterson and Gaven J. Watson. *Immunising CBC Mode Against Padding Oracle Attacks: A Formal Security Treatment*. In: *SCN 08*. Ed. by Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti. Vol. 5229. LNCS. Springer, Heidelberg, Sept. 2008, pp. 340–357 (cit. on p. 95).

[PW12] Kenneth G. Paterson and Gaven J. Watson. *Authenticated-Encryption with Padding: A Formal Security Treatment*. In: *Cryptography and Security: From Theory to Applications*. Ed. by David Naccache. Springer, 2012, pp. 83–107. URL: https://doi.org/10.1007/978-3-642-28368-0_9 (cit. on p. 96).

[PZ03] John Proos and Christof Zalka. "Shor's Discrete Logarithm Quantum Algorithm for Elliptic Curves". In: *Quantum Info. Comput.* 3.4 (July 2003), pp. 317–344 (cit. on p. 160).

[Rad11] Jay Radcliffe. *Hacking Medical Devices for Fun and Insulin: Breaking the Human SCADA System*. Black Hat USA. 2011. URL: https://media.blackhat.com/bh-us-11/Radcliffe/BH_US_11_Radcliffe_Hacking_Medical_Devices_WP.pdf (cit. on p. 5).

[Res18] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. 2018. URL: https://tools.ietf.org/html/rfc8446 (cit. on pp. 4, 13, 25, 101, 119, 173, 175).

[RG16] Andreea-Ina Radu and Flavio D. Garcia. *LeiA: A Lightweight Authentication Protocol for CAN*. In: *ESORICS 2016, Part II*. Ed. by Ioannis G. Askoxylakis, Sotiris Ioannidis, Sokratis K. Katsikas, and Catherine A. Meadows. Vol. 9879. LNCS. Springer, Heidelberg, Sept. 2016, pp. 283–300 (cit. on pp. 20, 22, 25).

[RKHP19] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Pöpper. *Breaking LTE on Layer Two*. In: *2019 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2019, pp. 1121–1136 (cit. on pp. 23, 25, 70).

[Rog02] Phillip Rogaway. *Authenticated-Encryption With Associated-Data*. In: *ACM CCS 2002*. Ed. by Vijayalakshmi Atluri. ACM Press, Nov. 2002, pp. 98–107 (cit. on p. 193).

[RS06] Phillip Rogaway and Thomas Shrimpton. *A Provable-Security Treatment of the Key-Wrap Problem*. In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Heidelberg, 2006, pp. 373–390 (cit. on pp. 136, 173).

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the Association for Computing Machinery* 21.2 (1978), pp. 120–126 (cit. on pp. 4, 175).

[RSWO17] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. *IoT Goes Nuclear: Creating a ZigBee Chain Reaction*. In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 195–212 (cit. on p. 5).

[Sch90] Claus-Peter Schnorr. *Efficient Identification and Signatures for Smart Cards*. In: *CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. LNCS. Springer, Heidelberg, Aug. 1990, pp. 239–252 (cit. on p. 8).

[Sho] Victor Shoup. *On Formal Models for Secure Key Exchange*. RZ 3120 (#93166), 19/04/1999 (cit. on p. 11).

[Sho04] Victor Shoup. *Sequences of games: a tool for taming complexity in security proofs*. Cryptology ePrint Archive, Report 2004/332. `http://eprint.iacr.org/2004/332`. 2004 (cit. on pp. 119, 153, 181).

[Sho94] Peter W. Shor. *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*. In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 124–134 (cit. on p. 160).

[Sho97] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM J. Comput.* 26.5 (1997), pp. 1484–1509. URL: `http://dx.doi.org/10.1137/S0097539795293172` (cit. on p. 8).

[Shr04] Tom Shrimpton. *A Characterization of Authenticated-Encryption as a Form of Chosen-Ciphertext Security*. Cryptology ePrint Archive, Report 2004/272. `http://eprint.iacr.org/2004/272`. 2004 (cit. on p. 193).

[Sig] *Signal*. URL: `https://signal.org/` (cit. on pp. 142, 143).

[Sig17a] Sigfox. *Secure SigFox Ready devices – Recommendation guide*. 2017 (cit. on pp. 22, 165).

[Sig17b] Sigfox. *SigFox Technical Overview*. 2017 (cit. on pp. 22, 165).

[SKH+02] James P. G. Sterbenz, Rajesh Krishnan, Regina Rosales Hain, Alden W. Jackson, David Levin, Ram Ramanathan, and John Zao. *Survivable Mobile Wireless Networks: Issues, Challenges, and Research Directions*. In: *Proceedings of the 1st ACM Workshop on Wireless Security – WiSE '02*. ACM, 2002, pp. 31–40. URL: `http://doi.acm.org/10.1145/570681.570685` (cit. on pp. x, 198).

[SLE+15] Nicolas Sornin, Miguel Luis, Thomas Eirich, Thorsten Kramp, and Olivier Hersent. *LoRaWAN Specification*. LoRa Alliance, version 1.0. 2015 (cit. on p. 74).

[SLE+16] Nicolas Sornin, Miguel Luis, Thomas Eirich, Thorsten Kramp, and Olivier Hersent. *LoRaWAN Specification*. LoRa Alliance, version 1.0.2. 2016. URL: `https://lora-alliance.org/resource-hub/lorawantm-specification-v102` (cit. on pp. 45, 46, 51, 54–56, 60).

[Sma16a] SmartCitiesWorld. *IoT connectivity for 100 million homes in China*. SmartCitiesWorld. 2016. URL: `https://smartcitiesworld.net/news/news/iot-connectivity-for-100-million-homes-in-china-1139` (cit. on p. 45).

[Sma16b] SmartCitiesWorld. *Semtech LoRa chosen for new IoT network in New Zealand*. SmartCitiesWorld. 2016. URL: https://smartcitiesworld.net/news/news/semtech-lora-chosen-for-new-iot-network-in-new-zealand-949 (cit. on p. 45).

[Sor17] Nicolas Sornin. *LoRaWAN 1.1 Specification*. LoRa Alliance, version 1.1, 11/10/2017. 2017. URL: https://lora-alliance.org/resource-hub/lorawantm-specification-v11 (cit. on pp. 22, 24, 25, 45, 63, 65, 71, 76, 77, 140, 165).

[SP05] Stefaan Seys and Bart Preneel. *Power Consumption Evaluation of Efficient Digital Signature Schemes for Low Power Devices*. In: *IEEE International Conference on Wireless And Mobile Computing, Networking And Communications*. Vol. 1. WiMob 2005. IEEE, 2005, pp. 79–86 (cit. on p. 172).

[SPLI06] JunHyuk Song, Radha Poovendran, Jicheol Lee, and Tetsua Iwata. *The AES-CMAC Algorithm*. RFC 4493. 2006. URL: https://tools.ietf.org/html/rfc4493 (cit. on p. 46).

[ST10] Yaron Sheffer and Hannes Tschofenig. *Internet Key Exchange Protocol Version 2 (IKEv2) – Session Resumption*. RFC 5723. 2010. URL: https://tools.ietf.org/html/rfc5723 (cit. on pp. 173, 175).

[ST16] Mohamed Sabt and Jacques Traoré. *Cryptanalysis of GlobalPlatform Secure Channel Protocols*. In: *Security Standardisation Research*. Ed. by Lidong Chen, David McGrew, and Chris Mitchell. Springer International Publishing, 2016, pp. 62–91 (cit. on pp. 23, 25).

[Stö17] Volker Stöhr. Personal communication. 2017 (cit. on p. 76).

[SZET08] Joseph Salowey, Hao Zhou, Pasi Eronen, and Hannes Tschofenig. *Transport Layer Security (TLS) Session Resumption without Server-Side State*. 2008. URL: https://tools.ietf.org/html/rfc5077 (cit. on pp. 173, 175).

[SIM18] SIMalliance. *SIMalliance Members Report Shipments of 4.9 Billion SIMs in 2017, while Estimated Total Available SIM Market in 2017 Increases 2.75% to 5.6 Billion Units*. 2018. URL: https://simalliance.org/media/press-releases/simalliance-members-report-shipments/ (cit. on pp. 82, 95).

[SSD] SSD Research Team. *OPAL library*. XLIM Labs, University of Limoges, France. URL: https://bitbucket.org/ssd/opal (cit. on p. 93).

[Sig16] Sigma Designs. *Security 2 Command Class, version 0.9*. 2016 (cit. on pp. 20, 25).

[Sil18] Silicon Labs. *Z-Wave Application Security Layer (S0)*. 2018 (cit. on pp. 20, 25).

[Tie18] Andrew Tierney. *Z-Shave. Exploiting Z-Wave downgrade attacks. Pen Test Partners*. 2018. URL: https://www.pentestpartners.com/security-blog/z-shave-exploiting-z-wave-downgrade-attacks/ (cit. on pp. 5, 20, 25).

[TM12] Joe-Kai Tsay and Stig F. Mjølsnes. *A Vulnerability in the UMTS and LTE Authentication and Key Agreement Protocols*. In: *Computer Network Security*. Ed. by Igor Kotenko and Victor Skormin. Berlin, Heidelberg: Springer, 2012, pp. 65–76 (cit. on pp. 23, 25).

[Tom19] Stefano Tomasin. Personal communication. 2019 (cit. on p. 75).

[TPG15] Hannes Tschofenig and Manuel Pegourie-Gonnard. *Performance of State-of-the-Art Cryptography on ARM-based Microprocessors*. NIST Lightweight Cryptography Workshop 2015 Session VII: Implementations & Performance. 2015. URL: https://csrc.nist.gov/csrc/media/events/lightweight-cryptography-workshop-2015/documents/presentations/session7-vincent.pdf (cit. on p. 7).

[Ttn] *The Things Network Stack V2*. URL: https://github.com/TheThingsNetwork/ttn/ (cit. on p. 55).

[TZV17] Stefano Tomasin, Simone Zulian, and Lorenzo Vangelista. *Security Analysis of LoRaWAN Join Procedure for Internet of Things Networks*. In: *IEEE Wireless Communications and Networking Conference Workshops*. WCNCW 2017. IEEE, 2017, pp. 1–6 (cit. on p. 74).

[Tex03] Texas instruments. *MSP430 Ultra-Low-Power MCUs*. 2003. URL: http://vega.elo.utfsm.cl/~lsb/elo325/aplicaciones/ApplicatioNotes/slab034f.pdf (cit. on p. 6).

[Unu] Roman Unuchek. *Dvmap: the first Android malware with code injection*. 08/06/2017. URL: https://securelist.com/dvmap-the-first-android-malware-with-code-injection/78648/ (cit. on p. 92).

[Uri10] Pascal Urien. *Introducing TLS-PSK authentication for EMV devices*. In: *2010 International Symposium on Collaborative Technologies and Systems, CTS 2010*. Ed. by Waleed W. Smari and William K. McQuay. IEEE, 2010, pp. 371–377. URL: https://doi.org/10.1109/CTS.2010.5478489 (cit. on p. 13).

[Vau02] Serge Vaudenay. *Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS...* In: *EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. LNCS. Springer, Heidelberg, 2002, pp. 534–546 (cit. on p. 95).

[VBMP17] Jo Van Bulck, Jan Tobias Mühlberg, and Frank Piessens. *VulCAN: Efficient Component Authentication and Software Isolation for Automotive Control Networks*. In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACSAC 2017. ACM, 2017, pp. 225–237. URL: http://doi.acm.org/10.1145/3134600.3134623 (cit. on p. 20).

[VHSV11] Anthony Van Herrewege, Dave Singelée, and Ingrid Verbauwhede. *CANAuth – A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus*. In: *ECRYPT Workshop on Lightweight Cryptography 2011*. 2011 (cit. on p. 20).

[VP16] Mathy Vanhoef and Frank Piessens. *Predicting, Decrypting, and Abusing WPA2/802.11 Group Keys*. In: *USENIX Security 2016*. Ed. by Thorsten Holz and Stefan Savage. USENIX Association, Aug. 2016, pp. 673–688 (cit. on p. 25).

[VP17] Mathy Vanhoef and Frank Piessens. *Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2*. In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, 2017, pp. 1313–1328 (cit. on pp. 20, 25).

[VP18] Mathy Vanhoef and Frank Piessens. *Release the Kraken: New KRACKs in the 802.11 Standard*. In: *ACM CCS 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM Press, Oct. 2018, pp. 299–314 (cit. on pp. 20, 25).

[Wei91] Mark Weiser. “The Computer for the 21st Century”. In: *Scientific American* 265.3 (1991), pp. 66–75 (cit. on p. 4).

[WGE+05] Arvinderpal S. Wander, Nils Gura, Hans Eberle, Vipul Gupta, and Sheueling Chang Shantz. *Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks*. In: *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*. PERCOM ’05. IEEE Computer Society, 2005, pp. 324–328. URL: http://dx.doi.org/10.1109/PERCOM.2005.18 (cit. on p. 172).

[Wor] LoRa Alliance Technical Committe Regional Parameters Workgroup (cit. on pp. 45, 53, 68, 69).

[Wu00] Thomas Wu. *The SRP Authentication and Key Exchange System*. RFC 2945. 2000 (cit. on p. 112).

[Yan17] Xueying Yang. *LoRaWAN: Vulnerability Analysis and Practical Exploitation*. 2017. URL: https://repository.tudelft.nl/islandora/object/uuid:87730790-6166-4424-9d82-8fe815733f1e/datastream/OBJ/download (cit. on p. 75).

[Yeg17] Alper Yegin. *LoRaWAN Backend Interfaces 1.0 Specification*. LoRa Alliance, version 1.0, 11/10/2017. 2017. URL: https://lora-alliance.org/resource-hub/lorawantm-back-end-interfaces-v10 (cit. on pp. 66, 70, 71, 77).

[YKPH11] Huihui Yap, Khoongming Khoo, Axel Poschmann, and Matt Henricksen. *EPCBC - A Block Cipher Suitable for Electronic Product Code Encryption*. In: *CANS 11*. Ed. by Dongdai Lin, Gene Tsudik, and Xiaoyun Wang. Vol. 7092. LNCS. Springer, Heidelberg, Dec. 2011, pp. 76–97 (cit. on p. 6).

[YL06a] Tatu Ylönen and Chris Lonvick. *The Secure Shell (SSH) Authentication Protocol*. RFC 4252. 2006. URL: https://tools.ietf.org/html/rfc4252 (cit. on pp. 4, 84).

[YL06b] Tatu Ylönen and Chris Lonvick. *The Secure Shell (SSH) Connection Protocol*. RFC 4254. 2006. URL: https://tools.ietf.org/html/rfc4254 (cit. on pp. 4, 84).

[YL06c] Tatu Ylönen and Chris Lonvick. *The Secure Shell (SSH) Protocol Architecture*. RFC 4251. 2006. URL: https://tools.ietf.org/html/rfc4251 (cit. on pp. 4, 84).

[YL06d] Tatu Ylönen and Chris Lonvick. *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253. 2006. URL: https://tools.ietf.org/html/rfc4253 (cit. on pp. 4, 84).

[ZF05] Muxiang Zhang and Yuguang Fang. “Security analysis and enhancements of 3GPP authentication and key agreement protocol”. In: *IEEE Transactions on Wireless Communications* 4.2 (2005), pp. 734–742 (cit. on pp. 23, 25).

[Zig14] ZigBee Alliance. *ZigBee specification*. 2014. URL: http://www.zigbee.org/download/standards-zigbee-specification/ (cit. on pp. 19, 25, 140).

List of Figures

List of Tables

List of Algorithms

LaTeX thesis template: https://github.com/rbost/thesis_template.

INSA de RENNES
Service des Formations

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse:
Secure Tunnels in Constrained Environments

Nom Prénom de l'auteur : FERREIRA LOIC

Membres du jury :
- Monsieur CANARD Sébastien
- Madame CHEVALIER Céline
- Madame FONTAINE Caroline
- Monsieur GILBERT Henri
- Madame NAYA-PLASENCIA Maria
- Monsieur POINTCHEVAL David
- Monsieur VAUDENAY Serge
- Monsieur AVOINE Gildas

Président du jury : Serge Vaudenay

Date de la soutenance : 18 Novembre 2019

Reproduction de la these soutenue

☒ Thèse pouvant être reproduite en l'état
☐ Thèse pouvant être reproduite après corrections suggérées

Fait à Rennes, le 18 Novembre 2019

Signature du président de jury

Le Directeur,

M'hamed DRISSI

INSTITUT NATIONAL des SCIENCES APPLIQUÉES - RENNES

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

INSA RENNES

Titre : Tunnels sécurisés pour environnements contraints

Mot clés : cryptanalyse, cryptographie à clé symétrique, protocole d'échange de clé authentifié, modèle de sécurité, Internet des Objets.

Resumé : Avec l'extension de l'Internet des Objets et l'usage croissant de terminaux à bas coût, de nombreux protocoles de sécurité sont déployés à grande échelle.

Cette thèse étudie le champ des protocoles d'échange de clé authentifié basés sur des fonctions cryptographiques symétriques. Nous montrons que les protocoles existants n'atteignent pas un niveau de sécurité correspondant à l'état de l'art en matière de protocoles cryptographiques. Nous décrivons des attaques pratiques contre deux tels protocoles actuellement utilisés. Nous présentons de nouveaux protocoles d'échange de clé entre deux et trois parties et décrivons des modèles de sécurité permettant de saisir leurs propriétés et de les analyser. Nos protocoles mettent uniquement en œuvre des fonctions symétriques. En même temps, ils garantissent des propriétés de sécurité plus fortes que les protocoles similaires. En particulier, ils garantissent la propriété de confidentialité persistante et permettent l'usage de reprises de session. Cela est particulièrement avantageux pour des terminaux disposant de peu de ressources en termes de calcul, de mémoire et d'énergie.

Title: Secure Tunnels for Constrained Environments

Keywords: Cryptanalysis, Symmetric-key cryptography, Authenticated key exchange, Security models, Internet of Things.

Abstract: With the rise of the Internet of Things and the growing popularity of constrained devices, several security protocols are widely deployed.

In this thesis, we investigate the field of authenticated key exchange protocols in the symmetric-key setting. We show that existing protocols do not achieve the most established levels of security properties, and describe practical attacks against two currently deployed protocols. We present new authenticated key exchange protocols for the 2-party and the 3-party cases, and describe suitable security models that allow capturing their security goals, and analysing them. Our protocols apply only symmetric-key functions. At the same time, they provide stronger security properties than comparable ones. In particular, they guarantee forward secrecy, and enable applying a session resumption procedure. This is particularly advantageous for low-resources devices with limited capabilities in terms of computation, memory, and energy.