On the security of instante messaging : towards solutions for multi-device and group applications

Céline Duguey

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1

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

Par
Céline DUGUEY

On the Security of Instant Messaging

Towards solutions for multi-device and group applications

Thèse présentée et soutenue à Rennes, le 14 décembre 2021
Unité de recherche : IRISA - UMR 6074

Rapporteurs avant soutenance :

Benjamin NGUYEN Professeur, INSA Centre Val de Loire
Damien VERGNAUD Professeur, Sorbonne Université

Composition du Jury :

Examinateurs :	Céline CHEVALIER	Maître de conférence, Université Panthéon-Assas Paris2
	Sylvain DUQUESNE	Professeur, Université de Rennes 1
	Benjamin NGUYEN	Professeur, INSA Centre Val de Loire
	Olivier SANDERS	Ingénieur de recherche, Orange Labs
	Damien VERGNAUD	Professeur, Sorbonne Université
Dir. de thèse :	Pierre-Alain FOUQUE	Professeur, Université de Rennes 1

Invité :

Julien DEVIGNE Ingénieur en cryptologie, DGA-MI

On the Security of Instant Messaging

Towards solutions for multi-device and group applications

Céline DUGUEY

Supervisor: Pierre-Alain FOUQUE

Remerciements

Ces quatre années de travail ont été rendues possibles grâce la confiance que beaucoup m'ont accordée. Ainsi, je tiens tout d'abord à exprimer ma gratitude à Pierre-Alain Fouque, qui a pris le temps de me recevoir dès 2016 et qui a accepté d'encadrer cette thèse pendant quatre ans. Nos discussions ont aiguillé mon travail tout en m'accompagnant vers plus d'autonomie. Julien Devigne a co-encadré ce travail côté DGA et m'a supportée comme co-bureau : merci de ta disponibilité lorsque j'interrompais inopinément tes autres activités pour te poser la question urgente du moment ou te faire part de mes doutes.

J'adresse mes remerciements chaleureux à Benjamin Nguyen et Damien Vergnaud qui m'ont fait l'honneur d'endosser le rôle de rapporteur, ainsi qu'à Céline Chevalier, Olivier Sanders et Sylvain Duquesne d'avoir accepté de faire partie de mon jury.

Merci à Sébastien Campion de sa disponibilité lors de nos travaux d'implémentation. Je garde un souvenir réjouissant de nos mercredis après-midi à tracer notre chemin dans le code de Signal.

J'ai pu mener mes travaux tout en ayant un poste à DGA-MI grâce à la confiance initiale de Raphaël Bost et Laurent Malaquin, ainsi qu'à la bienveillance de Yannick Landré et à la compréhension de Clément Gomez et Antoine Dallon, qui m'ont aidé à trouver un équilibre (parfois instable je le concède !) entre ces deux activités. J'ai également grandement bénéficié du soutien moral de l'ensemble de mes collègues de SCY grâce à l'ambiance toujours chaleureuse, positive et riche en sucre (et en beurre) de l'équipe : un grand merci à vous.

Mes mercredis (et/ou vendredis) à l'Irisa sont une parenthèse dédiée à la recherche toujours plaisante et je remercie l'ensemble des membres d'Emsec (Capsule/Spicy) de m'avoir intégrée malgré ma présence partielle. J'ai un souvenir particulièrement agréable et fructueux du groupe de travail mené avec Angèle Bossuat, Adina Nedelcu et Xavier Bultel. Le collectif était bienvenu pour oser approcher ces modèles de sécurité impressionnants à première vue. Je souhaite également remercier Katharina Boudgoust d'avoir accepté de partager avec moi le stress des derniers préparatifs techniques (*6) et Solène Moreau, car nos discussions en pauses café furent une véritable source d'apaisement personnel.

Je suis reconnaissante au Melting Note Orchestra et au projet Kolectiva de m'avoir apporté des respirations musicales bienfaisantes et d'avoir démontré la force du collectif, surtout en temps de confinement.

Merci à mes parents de leur soutien depuis le tout début. Merci à Sophie, Anna, Marie et Laurine : votre amitié indéfectible est un bien extrêmement précieux.

Julien, ta confiance en moi continuera toujours de m'étonner ! Merci de faire que tout soit possible si simplement. Elouen, Aénor, grâce à vous je peux mesurer l'ampleur de ce travail en centimètres (respectivement 27 et 37). À vos côtés j'apprends l'essentiel.

Résumé en Français

L'histoire de la cryptographie s'est principalement bâtie sur des guerres et des batailles de pouvoir. La science du secret a servi généraux et intrigants sur des générations : César et le code (élémentaire) qui porte désormais son nom, ou encore Marie Stuart, reine d'Écosse au milieu du XVI^{e} siècle, qui chiffrait ses messages pour fomenter le meurtre de sa rivale, Elizabeth première, reine d'Angleterre. Un des principes fondamentaux de la cryptographie a été énoncé dans le journal des sciences militaires, par Kerckhoffs (1883). Et lorsque la cryptographie est immortalisée au cinéma, c'est avec la machine Enigma, et son rôle pendant la seconde guerre mondiale. Une histoire bien sombre donc. Ils furent sûrement quelques uns, moins éminents, à s'intéresser à la cryptographie - et à sa contrepartie, la cryptanalyse - par curiosité ou, en tout cas, avec des desseins moins guerriers. Ceux là ne sont pas passés à la postérité. Mais l'avènement de l'informatique, d'Internet, et plus récemment encore des smartphones a totalement bouleversé l'usage de la cryptographie. Petit à petit, elle s'est introduite dans la vie civile, d'abord pour sécuriser des domaines considérés comme sensibles (les opérations bancaires par exemple). Puis elle est devenue accessible à tous ceux qui le souhaitent, via des applications comme Pretty Goog Privacy (PGP), non pas parce qu'ils ont quelque chose à cacher, mais sur le principe que, dans l'espace immensément ouvert que représente Internet, chacun a le droit à une vie privée. Les applications de messageries sécurisées, petites icônes sur nos smartphones, représentent l'apogée de cette ouverture au grand public, puisqu'elles mettent la cryptographie à disposition de tout un chacun, (presque) sans effort.

Pourquoi utilise-t-on Whatsapp ?

L'être humain est un animal social. Il a de tout temps éprouvé le besoin de communiquer. Il n'y a rien de surprenant, dès lors, que l'avènement d'un réseau mondial comme Internet ait vu l'émergence de nombreuses applications de communication : mail, chat, et plus récemment, grâce à la révolution des smartphones, messageries instantanées asynchrones. Whatsapp, FacebookMessenger, Telegram, Signal, Threema, Treebal, …, ces applications sont utilisées chaque jour par des milliards de gens sur l'ensemble de la planète. Tout comme les premiers chats sur ordinateur, elles sont libérées des latences que l'on retrouve dans les applications de mail et permettent de tenir des conversations presque en direct. Et contrairement aux premiers chats, la conversation ne s'arrête pas si l'un des protagonistes n'est plus en ligne. L'autre peut toujours continuer à déposer ses messages, qui seront vus plus tard.

De façon informelle, les messageries instantanées peuvent être vues comme un endroit où discuter avec ses amis, comme on le ferait dans la rue ou dans un café (avec ce bénéfice supplémentaire de n'être pas obligé d'être présent en même temps !). Assis à votre table, avec vos amis, vous n'apprécieriez que moyennement de sentir que les tables d'à côté ne perdent pas une miette de votre conversation. Vous choisiriez peut-être de partir, ou bien vous penseriez que « la prochaine fois, décidément, on choisira un autre endroit ! » Les messageries instantanées devraient suivre les mêmes règles, puisqu'on y tient les mêmes conversations. C'est l'objectif des messageries instantanées sécurisées (SIM), qui assurent une protection de bout en bout de nos conversations. Chaque message est chiffré sur le téléphone de l'émetteur, envoyé à qui de droit et ne peut être déchiffré que par le destinataire. Personne ne doit pouvoir s'immiscer dans la conversation, d'une quelconque façon. La démarche est vertueuse, mais il est intéressant de regarder de plus près si les utilisateurs sont vraiment sensibles à cet effort. Autrement dit, si l'endroit est sympa, sont-ils prêts à marcher plus longtemps pour trouver un café plus tranquille ?

Un cobaye nommé Johnny

La première personne qui peut nous aider à mieux comprendre l'intérêt du public pour la sécurité des messageries s'appelle Johnny. Il est le héros malgré lui d'une série d'articles traitant de l'ergonomie d'applications de sécurité. L'histoire de Johnny commence en 1999, dans un article intitulé « Pourquoi Johnny ne parvient-il pas à chiffrer ? » [WT99]. L'objectif de l'étude était d'identifier, dans le design d'une application de sécurité, les obstacles qui pouvaient empêcher un utilisateur - Johnny - de faire bon usage de l'application, voire le décourager de l'utiliser. Le postulat étant qu'une mauvaise utilisation d'une application censée apporter de la sécurité peut être plus dommageable qu'une mauvaise implémentation de l'application elle-même. L'étude porte sur PGP 5.0, un logiciel disponible dans le commerce, qui permettait à l'époque de chiffrer et de signer ses mails[1]. Le choix des auteurs n'est pas anodin : l'application est ouvertement destinée au grand public, puisque ses créateurs affirment qu'« avoir amélioré l'interface graphique côté utilisateur rend la cryptographie et ses mathématiques accessibles à un utilisateur d'ordinateur débutant. » L'idée est donc qu'il n'est pas nécessaire de comprendre les ressorts techniques pour utiliser le produit.

Certes le logiciel PGP 5.0 semble aujourd'hui un peu désuet. De plus, une différence majeure entre PGP et les messageries instantanées sécurisées (Secure Instant Messaging en anglais, SIM) est que le premier est installé par l'utilisateur dans le but de sécuriser ses communications, tandis que la cryptographie fait partie par défaut, par définition même, des messageries sécurisées. Ainsi certaines questions posées par l'étude, par exemple savoir si un utilisateur peut comprendre les bases

[1] L'application en est aujourd'hui à sa onzième version.

de l'authentification et du chiffrement, ou s'il peut utiliser correctement le logiciel après quelques heures d'efforts relatifs, ne sont pas applicables aux SIM. En effet, même si le public a montré un intérêt croissant pour la sécurité de ses communications, nous verrons un peu plus tard que ce n'est pas ce qui motive principalement le choix d'une application. Reste que certains constats obtenus d'une part par l'observation méticuleuse de l'interface graphique du logiciel, d'autre part via un test effectué auprès de douze « Johnny » en chair et en os, sont toujours pertinents. Par exemple, il est illusoire de penser que l'utilisateur lambda sera motivé pour se lancer dans la lecture de manuels ou même suivre des tutoriels, avant d'accéder à l'application. Mais l'enquête révèle également que le design ne peut pas suivre les règles habituelles. Dans le cas présent, PGP a beau être plutôt attractif (pour l'époque), avec des opérations simples (chiffrer, signer) accessibles par des boutons à cliquer clairement identifiés, seul quatre participants, pourtant tous habitués à envoyer des mails, a réussi à utiliser PGP 5.0 pour signer et chiffrer un message en une heure et demie. Par ailleurs, trois d'entre eux ont envoyé par mégarde le secret qu'ils étaient censés protéger. Aussi joli soit-il, un gros bouton « chiffrer » ne sert à rien si l'utilisateur ne comprend pas ce que signifie chiffrer. « Concevoir des applications de sécurité faciles d'usage pour des personnes qui ne comprennent pas vraiment la sécurité nécessite quelque chose en plus ».

Johnny a poursuivi sa carrière de cobaye dans d'autres travaux, mais nous le laissons là et, fort de nos premiers constats, nous allons nous intéresser de plus près au cas des messageries instantanées sécurisées.

Petite balade dans la tête d'un utilisateur

Depuis 1999 et le premier épisode de la saga Johnny, le scandale provoqué par la révélation en 2013 de programmes de surveillance de masse de la part des États-Unis ou encore du Royaume-Uni a eu un impact sur la perception qu'a le public des outils de communication numériques. C'est sûrement l'un des facteurs qui a permis l'émergence de nombreuses applications de messageries sécurisées, qui, pour la plupart, sont plus ergonomiques que leurs ancêtres. C'est ainsi qu'Unger *et al.* ont proposé, en 2015 ([UDB+15]), une revue d'une vingtaine de solutions existantes, en les comparant à la fois en terme de sécurité et d'ergonomie. On pourrait croire que, puisque les utilisateurs semblent de plus en plus sensibles à la sécurité de leurs communications et que, d'un autre côté, leur sont proposées de plus en plus d'applications de messagerie sécurisée ne demandant presque aucun effort, tous les problèmes sont résolus. Une étude récente, publiée par Dechand *et al.* ([DNDS19]) affirme au contraire que les choses ne sont pas si simples. Ces travaux montrent que l'ergonomie n'est pas l'unique obstacle auquel les concepteurs doivent faire face. Il semble qu'il faille aussi s'intéresser à ce qui se passe dans la tête de l'utilisateur : comment perçoit-il les risques liés à la sécurité ? Comment apprécie-t-il les solutions techniques apportées par la cryptographie, comme le chiffrement ou l'authentification ? En se basant sur la messagerie Whatsapp, les auteurs ont tenté de répondre à ces questions.

Whatsapp est une application de messagerie instantanée née en 2009. Son usage s'est massivement développé au cours des années 2010, succès « couronné » par son rachat par Facebook en 2014. En 2016, le chiffrement de bout en bout (du téléphone d'un usager jusqu'à celui de son correspondant) est adopté par l'application, qui le rend automatique. Dechan *et al.* ont vu dans cette annonce l'occasion d'interroger les usagers sur la façon dont ils percevaient la sécurité de Whatsapp. Ils ont comparé les résultats obtenus auprès de deux groupes, l'un interrogé en 2015, avant que Whatsapp ne devienne une messagerie sécurisée, l'autre en 2017, neuf mois après. Les deux groupes étaient pareillement composés de personnes ayant une compétence technologique dans la moyenne, avec un âge moyen autour de 30 ans. Les auteurs précisent par ailleurs que leur étude n'a été menée

qu'auprès de personnes de nationalité allemande et ne prétend donc pas être générale (des facteurs culturels peuvent évidemment entrer en jeu). Nous pensons néanmoins que ces résultats montrent une tendance intéressante.

Dis-moi où sont les amis de Johnny, je te dirai où il est

Le premier constat frappant est que, en 2015 comme en 2017, les participants avaient plutôt une bonne perception des différentes menaces qui pouvaient affecter la sécurité des communications numériques. Ils mentionnaient différents attaquants potentiels, du fournisseur de service aux agences de renseignements, en passant par les hackers et les sociétés commerciales. Pour ce qui est de Whatsapp plus particulièrement, il semble que le rachat par Facebook soit perçu comme une menace supplémentaire. Malgré tout, certains participants, bien que conscients des espions potentiels, pensaient que « les gens ordinaires sont peu susceptibles d'être visés par un programme de surveillance de masse. Selon eux, seules les personnalités riches, connues, les politiques ou encore les criminels étaient visés ». Ils se sentaient donc concernés, mais de loin. Ces résultats font écho à l'analyse quantitative menée par De Luca *et al.* ([DDO+16]), qui « suggère que l'influence des semblables est ce qui pousse principalement les gens à adopter une application de messagerie mobile plutôt qu'une autre, même pour les messageries sécurisées/privées et que la sécurité et le respect de la vie privée joue un rôle mineur. »

Peu de confiance envers le chiffrement

Le titre de l'article de Dechand *et al.* : « In encryption we don't trust », souligne bien le résultat principal de l'étude : bien que les participants arrivent à définir plus ou moins précisément le concept même de chiffrement, comme « une espèce de code secret » ou un « langage secret », certains mentionnant même l'usage d'une clé ou d'un mot de passe, tous admettent qu'ils ne pensent pas qu'il existe réellement une solution qui pourrait empêcher un attaquant un peu talentueux de casser le chiffrement. Dans le groupe interrogé en 2015, certains pensaient que Whatsapp était déjà chiffré, tandis qu'en 2017, tous n'étaient pas au courant que la messagerie avait adopté le chiffrement de bout en bout, et ce malgré le message affiché au début de toute conversation « les messages et les appels sont chiffrés de bout en bout. Aucun tiers, pas même Whatsapp, ne peut les lire ou les écouter. Appuyez pour en savoir plus. » Dans la grande majorité des cas, même une fois avertis de ce fait, les participants étaient convaincus que n'importe lequel des adversaires cités plus haut, ou au moins Whatsapp, pouvaient potentiellement lire leurs messages. L'une des conclusions des auteurs suite à ces constats est que les « usagers sont dépassés par la technologie en général et se considèrent sans recours et vulnérables face à des attaquants compétents ». Parmi les recommandations données à la fin de l'étude ressort donc le fait de parler le langage de l'usager. Par exemple, l'expression chiffrement de bout en bout était peu intelligible pour la plupart des participants. La réaction de l'un d'entre eux est révélatrice : « oh, voilà donc de quoi parlait cette notification ennuyeuse ».

Pourquoi s'authentifier ?

Dans l'univers de la cryptographie, l'authentification vise à s'assurer que l'on parle bien avec la personne avec qui l'on est censé parler. Si cette définition est claire pour les cryptographes, elle l'est beaucoup moins pour les non spécialistes. Les participants de l'étude qui nous intéresse, que ce soit en 2015 ou en 2017, ne comprenaient ni comment on pouvait parvenir à s'authentifier, ni à quoi cela servait. L'étude met ainsi en avant des réactions représentatives. Premièrement, la conviction que l'on peut se fier à la notion de compte personnel et aux identifiants pour savoir à

qui l'on parle : « Supposons que moi et mon ami ayons chacun un compte, Alex27 et Katie07 par exemple. Je lui envoie un message. Pourquoi Pia23 pourrait-elle le lire ? ». Le fait que Pia23 puisse se jouer de lui en se faisant passer pour Alex27 ne constitue pas une menace. Deuxièmement, le sentiment qu'« ils se rendraient bien compte si un message ne provenait pas du bon émetteur (par des différences de style, de langage) ». Ces constats sur l'authentification prennent toute leur importance si l'on considère Whatsapp, car l'authentification est la seule étape qui nécessite l'intervention de l'usager. Pour identifier formellement le propriétaire d'un compte, chaque utilisateur devrait, avant de communiquer avec cette personne, échanger une valeur (l'empreinte cryptographique de la clé publique correspondante), par un canal tiers, par téléphone ou en scannant un QR code en direct par exemple. Les utilisateurs sont avertis que cette action devrait être faite, mais elle reste néanmoins optionnelle. Aucun participant (sauf un) n'avait connaissance de ce code de vérification. Et l'un d'entre eux a ouvertement affirmé que « ce serait trop peu pratique de scanner les QR codes de tous les contacts. » Ce n'est pas étonnant au vu des précédents constats sur l'authentification. Mais cela confirme la nécessité d'imaginer des systèmes qui ne requièrent aucune action de l'utilisateur : « les utilisateurs ne devraient pas avoir à se soucier de la sécurité - elle devrait juste être là pour eux ».

Mes contributions

Dans cette thèse, nous nous intéressons à deux services proposés par les messageries instantanées : la connexion de plusieurs appareils et les communications de groupe. S'ils peuvent être considérés comme des options, les résultats précédents nous montrent bien que ce n'est pas la sécurité qui va guider les utilisateurs vers une application et, à choisir, ces services seront sûrement préférés à l'assurance d'un chiffrement bout en bout. C'est pourquoi toutes les applications, sécurisées comprises, les proposent. Tout l'enjeu, et c'est l'objet des travaux présentés ici, consiste à parvenir, pour ces services, à un niveau de sécurité optimal, comme pour les communications deux à deux classiques. En gardant à l'esprit que la cryptographie ne constitue peut-être pas la plus grande difficulté.

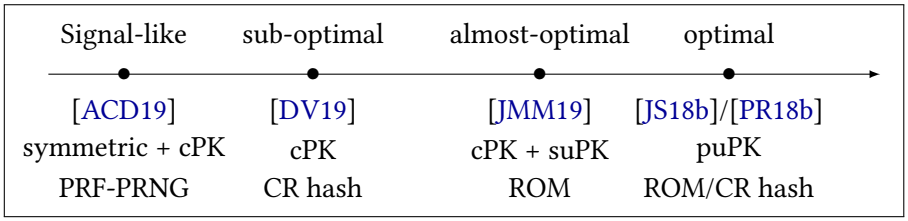
Ce manuscrit s'organise autour de deux contributions principales : la première, sur le multi appareil, est écrite avec Sébatien Campion, Julien Devigne et Pierre-Alain Fouque et a fait l'objet d'une présentation à ACNS 2020 ([CDDF20]). La seconde correspond à des travaux menés avec Julien Devigne et Pierre-Alain Fouque, présentés à ESORICS 2021. La présentation de ces contributions est précédée d'un état de l'art détaillé de la littérature concernant les échanges de clés à cliquet (ratcheted key exchange), aspect intéressant selon nous car ce concept, qui est au coeur de nombreuses messageries sécurisées, est plutôt récent, et a fait l'objet d'un nombre non négligeable de publications ces cinq dernières années.

Un état de l'art sur les échanges de clés à cliquet

En 2013, l'application de messagerie TextSecure, rebaptisée Signal peu après, introduisait un nouveau protocole pour sécuriser les échanges entre deux interlocuteurs[2]. Dès l'origine, les implémentations des algorithmes sont accessibles en open source, et trois ans plus tard, Moxie Marlinspike (à l'origine de la fondation qui gère Signal) et Trevor Perrin publient les détails de ce qui fait la sécurité de Signal : X3DH et l'algorithme intitulé Double Ratchet. X3DH est un protocole permettant à deux participants, Alice et Bob, de négocier un secret commun, sans nécessairement être en ligne

[2]Initialement, TextSecure était un protocole de chiffrement des SMS/MMS basé sur Off-the-Record. En 2013, l'application a migré du réseau GSM vers un transport data (reposant donc sur Internet et les couches TCP/IP), et a fait évoluer son protocole d'authentification et de chiffrement en parallèle.

même temps. Ce secret est usuellement utilisé par un protocole de chiffrement, qui va assurer la confidentialité des messages échangés. Le Double Ratchet va plus loin car il permet à Alice et Bob, une fois qu'ils partagent un secret de base, de rafraîchir régulièrement ce secret et d'en dériver de nouvelles clés cryptographiques, qui serviront à sécuriser leurs communications. Ce qui fait l'intérêt du Double Ratchet, c'est qu'il garantit la sécurité des messages (des clés) passés et futurs même si un adversaire accède aux clés long termes identifiant les deux protagonistes, ou si cet adversaire apprend les secrets présents mais reste passif. Il doit corrompre les clés régulièrement s'il veut suivre la conversation. Ces caractéristiques sont appelées confidentialité persistante (forward secrecy) pour la première et confidentialité future ou rétablissement pour la seconde. Et ce sont notamment ces caractéristiques qui ont attiré l'attention sur ce nouveau type d'échange de clés évolutif, baptisé échange de clés à cliquet. Après l'analyse de la sécurité de Signal en 2017 par Cohn Gordon *et al.* [CCD+17], Bellare *et al.* ont proposé en 2018 une formalisation des échanges de clés à cliquet [BSJ+17]. Dans ce premier effort, seul l'un des deux participants à l'échange fait évoluer le secret (enclenche le cliquet). Parmi les articles qui ont fait suite à ce travail fondateur, nous en retenons cinq, dont nous considérons qu'ils sont représentatifs. Les travaux de Poettering et Rösler [PR18b] ainsi que ceux de Jaeger et Stepanovs [JS18b], tous deux parus à Crypto 2018, étudient la sécurité maximale que l'on peut attendre de ce genre d'échange de clés, que ce soit au niveau de la confidentialité persistante ou future. L'intérêt théorique de leurs propositions est indéniable mais elles nécessitent des primitives de cryptographie à clé publique permettant une mise à jour publique des clés, et les constructions pratiques qu'ils proposent pour se conformer à ce modèle idéal font appel à du chiffrement basé sur l'identité hiérarchique, une brique cryptographique qui n'est, à l'heure actuelle, pas assez efficace pour ce genre d'application. L'année suivante, Jost, Mauerer et Mularczyk [JMM19], Durak et Vaudenay [DV19] et Alwen Corretti et Dodis [ACD19] ont également proposé des modèles de sécurité permettant d'atteindre différents compromis entre sécurité et efficacité. Les derniers formalisent une sécurité proche de celle proposée par Signal, dont on sait qu'elle est atteignable en pratique, tandis que les premiers définissent une sécurité presque optimale, mais ne proposent pas d'implémentation. La figure ci-dessous résume le classement des différentes propositions sur le plan de la sécurité.



Positionnement des différentes propositions en terme de sécurité. cPK signifie cryptographie à clé publique classique, suPK cryptographie à clé publique avec mise à jour secrète, et puPK cryptographie à clé publique avec mise à jour publique. Nous précisons également si les preuves de sécurité associées sont données dans le modèle classique, en se basant sur des fonctions de hachage résitantes aux collisions (CR) ou dans le modèle de l'oracle aléatoire (ROM).

Plusieurs appareils pour un utilisateur

Une caractéristique ressort de l'état des lieux précédent : les échanges de clés à cliquet sont par défaut définis pour deux participants, ce qui représente, dans le cas d'une messagerie, deux appareils, généralement le téléphone de Bob et celui d'Alice. Or la tendance est actuellement à la multiplication

des appareils : smartphones, tablettes, ordinateurs portables, montres, ... Pour que cette pluralité ait un sens (du moins sur le plan technologique), le minimum attendu est que ces appareils puissent inter-opérer. Si l'on parle de messagerie, un utilisateur doit pouvoir avoir un compte unique et suivre ses conversations depuis l'un ou l'autre de ses appareils. Pourtant, aucun protocole adressant spécifiquement cette problématique n'a été proposé ni analysé jusqu'ici. Les équipes de la fondation Signal ont publié un protocole baptisé Sesame, qui répond à cet usage en multipliant les canaux Signal traditionnels, deux à deux, entre tous les appareils qui entrent en jeu dans la conversation : tous ceux de Bob et d'Alice. Un inconvénient, important selon nous, de cette solution, outre que le fait que le nombre de canaux deux à deux nécessaires augmente de façon polynomiale avec les appareils d'Alice et de Bob, est que chacun doit voir les appareils de l'autre pour pouvoir établir un canal avec eux.

Nous proposons une solution alternative, qui se décompose en deux parties : la première est un mécanisme de multicast dont les clés évoluent dans le temps, qui permet à un groupe d'appareils de communiquer entre eux tout en conservant les propriétés de sécurité de Signal, dont la confidentialité persistante et future. Cette nouvelle primitive, que nous appelons multicast dynamique à cliquet est proche des multicasts existant, mais prend en compte des contraintes supplémentaires. L'une est qu'il ne doit pas y avoir d'autorité centrale, car aucun appareil ne doit être maître. L'autre est le besoin de rafraîchir régulierement les clés. Enfin, nous utilisons le fait que les appareils qui doivent communiquer appartiennent à un utilisateur unique, ce qui permet d'assurer une authentification initiale forte plus simplement. Nos travaux définissent un modèle de sécurité associé à cette primitive, qui formalise quelles propriétés sont attendues et sous quelles conditions. L'un des points d'intérêt de ce modèle réside dans la définition des sessions correspondantes, une session étant l'exécution du protocole par un participant. Cette notion est usuelle pour les échanges de clés, et sert à identifier les deux sessions qui exécutent le protocole ensemble, mais il a fallu l'étendre pour prendre en compte plus de deux participants. Comme les sessions ont des durées de vie longues (c'est une des caractéristiques des sessions de messagerie asynchrone) et que les appareils peuvent être ajoutés ou révoqués en cours de session (c'est l'aspect dynamique du protocole), il a également fallu définir la notion de correspondance entre sessions relativement aux différentes étapes du protocole. Nous avons donc introduit la notion de chaînes de session, qui permettent de relier, dans l'analyse de sécurité, deux appareils qui ont participé à la même instance du protocole mais qui étaient présents à des étapes différentes. Enfin nous proposons une construction basée sur des primitives cryptographiques classiques : un chiffrement à clé publique et un code d'authentification de message (MAC).

La seconde partie de ces travaux présente une définition ainsi qu'un modèle de sécurité pour un échange de clés à cliquet pour plusieurs appareils. Une attention toute particulière est portée à la définition de la fraîcheur d'une session, qui encadre en quelque sorte les pouvoirs donnés à l'adversaire que l'on considère, afin de pouvoir transposer les conditions de fraîcheur du modèle pour la version classique de Signal proposé par Cohn Gordon *et al.* ([CCD+17]). Encore une fois, la notion de correspondance des sessions requiert de la vigilance et de nouvelles définitions. Nous montrons ensuite qu'il est possible de composer le multicast à cliquet défini plus haut et le protocole Signal mentionné dans l'état des lieux, pour obtenir un protocole d'échange de clés à cliquet à plusieurs participants, dont nous prouvons qu'il est sûr dans le modèle défini ci-avant.

Sécuriser le (futur ?) protocole de messagerie de groupe MLS

Depuis février 2018, des chercheurs et des ingénieurs travaillent ensemble dans le cadre d'un groupe de travail de l'IETF (Internet Engineering Task Force) afin de mettre au point une solution de messagerie de groupe asynchrone sécurisée, baptisée MLS, pour Messaging Layer Security. Cette

solution vise des propriétés de sécurité semblables à celles de Signal, à savoir la confidentialité et l'authentification propres à un canal sécurisé, ainsi que la confidentialité persistante et future. Le coeur de leur proposition est un protocole baptisé TreeKem. Comme son nom l'indique (Tree signifie arbre en anglais), ce protocole est basé sur une structure d'arbre binaire et permet à n'importe quel membre du groupe - une feuille de l'arbre - de mettre à jour le secret du groupe : la racine. Nos travaux s'intéressent à une faiblesse du protocole, identifiée formellement dans le dernier avant-projet rendu public (draft 11 [BBM+20]) : lorsqu'un utilisateur met à jour le secret racine, il doit envoyer des informations à certains nœuds de l'arbre, afin que chaque membre du groupe - chaque feuille - puisse prendre en compte cette mise à jour (chaque feuille récupère l'information auprès de son nœud plus proche parent). Chaque nœud reçoit donc un secret qui lui est propre et qui dépend de sa position dans l'arbre, mais tous ces secrets sont néanmoins dérivés d'un même secret initial choisi par l'auteur de la mise à jour. De chacun de ces secrets est également dérivée une information publique, envoyée à tous. Dans la version actuelle, les participants ne peuvent vérifier la validité de la mise à jour reçue qu'une fois qu'ils l'ont déchiffrée. Si un utilisateur malveillant envoie des informations de mise à jour incorrectes à une certaine partie de l'arbre, les participants correspondants ne vont le voir qu'à réception. Et ceux qui auront reçu des informations correctes prendront en compte la mise à jour, ce qui exclura d'office les premiers. Nous introduisons une étape de vérification effectuée par le serveur, avant de délivrer les messages de mise à jour. L'idée étant que le serveur puisse vérifier que tous les messages chiffrés de mise à jour contiennent des informations correctes, sans rien apprendre d'autre sur le contenu des différents messages, qui est secret puisqu'il mène à la clé de groupe. Notre solution s'appuie sur deux primitives cryptographiques : un protocole de preuve de connaissance à divulgation nulle d'une part, qui montre que les données publiques de la mise à jour sont bien toutes calculées à partir de secrets dérivés d'un secret initial commun. Du chiffrement vérifiable d'autre part, qui permet à l'auteur de la mise à jour de prouver au serveur que les éléments chiffrés à destination des autres nœuds sont bien les secrets qui sont considérés dans la preuve de connaissance.

Les clés publiques de chaque nœud sont obtenues en appliquant successivement la fonction de dérivation HKDF au secret initial, puis en effectuant une multiplication dans un groupe fini d'ordre premier (instancié par une courbe elliptique). Malgré l'intérêt fort de la communauté cryptographique pour les protocoles de preuve à divulgation nulle, peu de solutions existent pour prouver l'exactitude de calculs qui marient des étapes algébriques et des étapes représentées plus efficacement par un circuit. Nous proposons donc deux protocoles distincts pour calculer une preuve de connaissance à divulgation nulle qu'un élément y est bien le résultat du calcul d'un circuit C sur une entrée x, étant donné uniquement les mises en gage algébrique (de type Pedersen) de l'entrée x et de la sortie y. Le premier est une extension direct du travail de Backes *et al.* ([BHH+19]), basé sur la technique de calcul multipartite « dans la tête » (MPC in the head). Le second (décliné en deux variantes) est réservé au cas ou le circuit C représente une fonction pseudo aléatoire car il s'appuie sur des propriétés spécifiques de ces fonctions.

Contents

1 Introduction

In its early stages, cryptography was a matter of war and power: Ceasar the emperor and the cipher that bear his name, has become a classical example, or Mary Stuart, queen of Scotland, that encoded her messages sent from her captivity in the mid 16th century, to foment the murder of Elizabeth the first, back then Queen of England. Even the basics of cryptography, were originally edited in a military manual by Kerchoffs in 1883. And when cryptography is immortalized in a box-office success, it is with the famous machine Enigma and its role during the second world war. There is no doubt that some common people, beyond those examples that have gone down to posterity, got interested in cryptography, as a hobby or, at least, without warlike intentions. But one can bet they were few. The advent of computer science and, later, of the Internet has completely rescaled the role of cryptography. Little by little, it has been introduced to secure specific sensitive activities, in the domain of banking for instance, before being proposed directly to secure one-to-one communications, for anyone willing to, with applications such as OpenPGP. In that sense, Secure Instant Messaging (SIM) solutions represent a kind of apotheosis: they offer cryptography for anyone in his everyday life communications. In the first part of this introduction, we will get interested in how "anyone" welcomes and understands this offer. In the second part, we will detail our contributions to the domain of SIM.

Contents

1.1 Why Johnny uses Whatsapp ?

Humans are social beings. One of our primal need is to communicate with each other. Hence, there is no surprise that, with the advent of the Internet, a large number of communication solutions appeared, among them mail, Instant Messaging and more recently with the revolution of smartphones, asynchronous Instant Messaging applications. WhatsApp, FacebookMessenger, Telegram, Signal, Threema, Treebal[1],..., are daily used by billions of people around the world. Instant messaging can be seen as a place to converse informally with a friend, as you would do if you'd meet him in the street, or in a *café*. But sitting at your table, you would feel pretty uncomfortable if you knew that the other customers around you were listening to your conversation. You would maybe whisper or leave and find another quieter place. Messaging applications follow the same rules: they should provide some privacy to their users. This is the purpose of Secure Instant Messaging apps, that propose end-to-end security. Authentication and privacy are offered from the sender's phone, to the receiver's phone, and should not be defeated in between. But an interesting question is how much people really care for virtual privacy. And how long they are eager to walk to find quieter *café* !

1.1.1 Usability: come and meet Johnny

To get some clues, one can ask Johnny. Johnny is the reluctant hero of a series of works about security and usability. Johnny's story begins in 1999, in the episode "Why Johnny can't encrypt" by Whitten and Tygar ([WT99]). The purpose was to evaluate the design of security interfaces, to identify what where the obstacles that refrain a user - Johnny - from correctly using them (or, even more, from using them at all). The assessment was that a wrong use of a security application could cause as much - or even more - damage than a poor implementation. The authors conducted two parallel studies on PGP 5.0[2]. This commercially available interface came over a mail provider, to secure the communications by encrypting and signing (electronic signature) the emails. It openly targeted a broad public, as its creators stated for instance that "significantly improved graphical user interface makes complex mathematical cryptography accessible for novice computer users." No need to understand the technical principle to use the product. And there should be no need to worry about it. How many car drivers really understand how their motor works? But a driver needs to learn how to properly use his car, because this is not intuitive. If no such training is available for an application, then the use must be intuitive. Whitten and Tygar defined more precise criteria for a secure interface to be usable in practice.

Definition 1.1 (Security software usability)**.** *A security software is usable if the people who are expected to use it:*

- *Are reliably made aware of the security tasks they need to perform*
- *Are able to figure out how to successfully perform those tasks*
- *Don't make dangerous errors*
- *Are sufficiently comfortable with the interface to continue using it*

We will not go into the details of this study, as the interface of PGP 5.0 is no longer relevant. Also a main difference with a Secure Instant Messaging application, is that this software was to

[1] This one is not so famous yet but has been developed in Rennes.
[2] The application is now in version 11.

be installed by people willing to protect their communications, while SIM (not all but most of them) provide end-to-end security by default. Among the questions Whitten and Tygar wanted to explore were whether a person - our Johnny - who acquires PGP would understand the basics of cryptography (for instance that privacy is achieved by encryption, or that public keys need to be exchanged), and also whether he would be able to use correctly PGP "within a few hours of reasonably motivated effort". In both cases, this is already a lot to ask to a user, and does not fit the profil of SIM users. Even if the need for more secure means of communication increases, we will see later on that the billions of WhatsApp users do not raise security as a first motivation for choosing this application. However some assessments, obtained from theoretical observations of PGP and a case-study run with twelve participants, are still up-to-date. The design for instance, should not assume that users will be motivated to read manuals, or follow tutorials. Maybe more meaningful is the comment that "despite the fact that PGP 5.0 is attractive, with basic operations neatly represented by buttons with labels and icons" only one-third of the participants, while being generally educated and trained experienced at using email, were able to use PGP 5.0 to correctly sign and encrypt an email message within 90 minutes, and one-quarter of them accidentally exposed the secret they were meant to protect. The conclusion is that a well designed encrypt button is useless if the user does not understand what encrypting corresponds to. The classical conception rules do not apply, because the concept behind the buttons are not familiar to the user, and the authors conclude that "designing security that is usable enough to be effective for those who don't already understand it must thus require something more."

New episodes of Johnny's adventures are regularly released ([GM05], [CGM+11] and some more) but we will leave our hero there, to focus on the case of messaging, keeping in mind that offering cryptography to the masses is challenging.

1.1.2 A short trip in the user's mind

From the time we quit Johnny, the scandal of the American mass surveillance in 2013 has clearly modified the use of communication technology. Or rather, has changed the perception of the communication technologies by their users. This motivated the deployment of numerous secure messaging applications, which, for most of them, seem to provide a better usability than their predecessors. Unger *et al.* provide in [UDB+15] a large overview of the different solutions available in 2015 and compare those offers in terms of security and usability. Their work will follow us in the next chapters of this thesis. An intuitive thought would tell us that, if on the one side people are more and more aware of privacy concepts and, on the other side, if more applications (pretend to) offer such privacy with nearly no effort, everyone should be happy. A recent study shows that it is not that simple. In [DNDS19], Dechand *et al.* tell us that the usability problems are not the only obstacle to the adoption of encryption solutions. The mental model of the users are to be considered. How aware are they about security threats? How do they receive technical solutions such as encryption and authentication? The authors focus their study on messaging applications and their main tool is WhatsApp.

WhatsApp is an Instant Messaging application that was released in 2009, and that saw its adoption massively raise along the 2010's, before being bought out by Facebook in 2014. In 2016, WhatsApp announced that it introduces end-to-end encryption as an automatic feature. Dechan *et al.* took advantage of this announce to study how people welcome the introduction of security features in their every day life means of communication. They interviewed two sets of users, at different times: the first set was questioned in 2015, before WhatsApp's announcement. The second set was interviewed in 2017, nine months after end-to-end encryption had been introduced. The sets

were composed with various profile, with an average age around 30 in both cases and a technical knowledge estimated as medium. The authors underline that their study was conducted only with German participants, and so, does not pretend to be applied to other cultures directly. We feel that this provides a general tendency, even if it would benefit from some geographical enlargement.

Tell me where Johnny's friends are, I'll tell you where he is. Firstly, it is interesting to note that the participants, both in 2015 and 2017, were aware that security threats exist. They mentioned mobile providers, governments, intelligence agencies, hackers and commercial companies as being potential spies. Considering WhatsApp, the fusion with Facebook appears as an additional threat. However, some also believed that "ordinary people are not likely to be targeted for surveillance. They stated that either rich, famous people, politicians or criminals are targeted." They do not feel directly affected. These results echo with the quantitative analysis conducted by De Luca *et al.* ([DDO+16]), that suggests "that peer influence is what primarily drives people to use a particular mobile IM, even for secure/private IMs, and that security and privacy play minor roles."

In encryption we don't trust. The title of the article enlightens the main result. While the participants could explain the idea of encryption, as being "a kind of secret code" or "secret language", some of them mentioning the need for a key or a password, they do not believe that there really exists a solution stopping skilled attackers from breaking encryption. Considering WhatsApp, while some of the participants of 2015 thought it was already encrypted, not all of the participants of 2017 were aware of the novel end-to-end encryption feature, despite the notification message that appears at the beginning of every conversation: "Messages you send to this chat and calls are now secured with end-to-end encryption. Tap for more info." And, in nearly every case, even when informed of it, they were convinced that potentiality any of the above cited adversaries, or at least WhatsApp, could read their messages: if WhatsApp controls the algorithm, then it can access to all the data. One of the conclusion of the authors is that "users are overwhelmed by technology in general and consider themselves to be helpless and vulnerable against skilled attackers". Among the recommendations, one is to speak user's language. In fact, the terms "end-to-end encryption" were opaque for most of the participants. The reaction of one participant when asked about WhatsApp end-to-end encryption speaks for itself:"oh that was what this annoying notification was about".

Why authenticate? In terms of cryptography, authentication deals with being sure that the person you are exchanging with is the intended person. If this concept seems quiet clear for cryptographers, it reveals to be more foggy for non specialists. The participants, both in 2015 and 2017, do not understand how authentication can be achieved, nor why it is necessary. The study highlights representative behaviours: firstly, the conviction that one can rely on personal account and identifiers to know who you speak to "I assume that my friend and I have accounts, Alex27 and Katie07, for example. Then I send a message to them. Why should Pia23 read along?" It does not appear as a threat to this participant that Pia23 could fool him by making him believe that she is Alex27 for instance. Secondly, the feeling that "they would notice if a certain message does not come from the sender (different style of writing or language)." This review about authentication is of prime importance considering WhatsApp, as authentication is the only step that requires a move from the user. In order to identify an account owner, one should, before starting a conversation with him, exchange a QR code or a sequence of numbers (which are the peer cryptographic key fingerprint) via another channel (direct meeting, phone...). This step is notified to users, but remains optional. All participants but one did not know about this security code verification. And one clearly

expressed that “it would be too inconvenient to scan QR codes of all contacts”. This confirms the demand for user-interaction-free solutions: “users should not have to care about security - it should just be there for them.”

1.2 Contributions

In this thesis, we are interested in two services proposed by messaging applications: the multi device accessibility and group messaging. If they can be considered as options, the above results show that security will not be the main motivation for a user to select his favorite application and, in his choice process, these options will surely be prefered to a certified end-to-end security. Hence, nearly all messaging applications, secure ones included, are willing to offer them. What is at stake - and this is the purpose of the work presented here, is to reach, for those services, an optimal level of security, as for the classial one-to-one communications. While keeping in mind that the difficulty of providing security does not only lie in the cryptographic corners. This manuscript exposes two main contributions: the first [CDDF20], co authored with Sebastien Campion, Julien Devigne and Pierre-Alain Fouque, was presented (virtually) at ACNS 2020. The second contribution [DDF21], with co-authors Julien Devigne and Pierre-Alain Fouque was presented (still virtually) at ESORICS 2021. We adjoin to these contributions an expanded state-of-the art of the literature about Ratcheted Key Exchange, which we think of interest for this concept, which is at the heart of many secure messaging apps, is quiet novel and a wide range of papers have been published in a short period of time.

1.2.1 A survey of the Ratcheted Key Exchange literature

In 2013, TextSecure introduced a new key management system for secure messaging. Some years later, TextSecure was renamed Signal and in 2016, the two algorithms that composes this key manadgement: X3DH and the Double Ratchet algorithm, were publicly released by their designers, Trevor Perrin and Moxie Marlinspike, in the white papers [MP16a] and [MP16b]. X3DH is a non interactive key exchange, that enables two participants, Alice and Bob, to negociate a shared secret without being online at the same time. The Double Ratchet comes over and allows Alice and Bob to refresh their shared secret regularly and derive cryptographic keys from it, keys that will be used to secure their communications. One interesting property of the Double Ratchet is that the confidentiality of past and future messages is still guaranteed even after an exposure of long-term keys or even of state secrets by a passive adversary. This forces the adversary to expose keys regularly. Those features are often identified as forward secrecy and healing. Because of those feature, the literature got highly interested in this new kind of evolving key exchange, identified as Ratcheted Key Exchange.

We propose, in chapter 3, a review of this literature. We first recall the advent of the ratchet mechanism, going from the historical Off-the-Record messaging protocol to our subject of interest: the Double Ratchet and Signal. Then, we focus mainly on five major articles that we consider as representative (Poettering and Rösler [PR18b], Jaeger and Stepanovs [JS18b], both published at Crypto 2018, Durak and Vaudenay [DV19] published at IWESEC 2019, Jost, Maurer and Mularczyk [JMM19] and Alwen, Coretti and Dodis [ACD19] both published at Eurocrypt 2019), trying to finely understand the differences, in term of security, between those propositions.

While comparing the above works, we try to highlight the reasons why the most optimal solutions require complex cryptographic primitives such as identity based encryption, secure but yet unaffordable for a practical messaging application.

1.2.2 A multi device solution for Signal

Our survey about Ratcheted Key Exchange underlines that these protocols are designed for one-to-one communications only. But the technological race multiplies the number of devices that each user can access: smartphones of course, laptops, pads, watches... For this plurality to have a sense (at least at a technological point of view), the minimal usability requires that these devices should be able to inter operate. Considering messaging applications, a user shall have a single account accessible from any of his devices. We noticed, when we started looking in that direction, that no specific protocol had been proposed, nor analysed. The Signal Team has released a protocol called Sesame, that only duplicates one-to-one Signal channel between any pair of devices present in a conversation. A main drawback of this solution, in addition to the non scalability of this multiplication of one-to-one channels, is the fact that Alice can identify all of Bob's devices.

We propose an alternative solution that overcomes this obstacle. Our contribution comes in two parts: in a first step, we propose a ratcheted multicast that enables a bunch of devices to communicate together while keeping the security properties of the Signal messaging, among them PFS and PCS. This new primitive, identified as a Ratcheted Dynamic Multicast (RDM), is close to existing multicast protocols, with more constraints: no central authority, a regular renewal of the keys, and some specific features: we take advantage of the fact that the devices belong to a single user for the initial authentication process. We provide a security model for this new primitive. One point of interest is a specific definition of matching sessions. Firstly, the traditional of matching has to be extended to more than two sessions. Secondly, because the sessions are long-lived and continuously evolving ones, the matching has to be defined relatively to the evolution of the shared state. We notably introduce a concept of chained sessions to link sessions that run on different devices at different times but correspond to a same execution. Finally, we propose a construction based on traditional cryptographic primitives.

In a second step, we define a security model for a multi device version of the Signal protocol, which is formalized as a multi stage key exchange. We focus on the definition of the freshness conditions, that are meticulously transposed from the one-to-one to the multi device version. Again in this case, the definition of the matching requires a peculiar attention. We then detail how our RDM can be associated with the existing Signal, with minor modifications, to provide a multi-device ratcheted key exchange, that allows for a single Signal channel between the two participants, no matter how many devices they use. We prove this modular approach to be secure in the sense of the above model. Both our security model and our proof are settled on the original analysis of Signal given by Cohn-Gordon *et al.* in [CCD+17].

1.2.3 Improving the security of the -to be standardized- Messaging Layer Security group messaging protocol

Since february 2018, a group of researchers and engineers[3] is working within the Internet Engineering Task Force (IETF) to design a secure group messaging solution. Their protocol is called MLS: Messaging Layer Security. As the one-to-one Signal, its aims at providing forward and post compromise secrecy. The cryptographic heart of MLS is called TreeKEM. As the name suggests, this protocol is based on tree structure and enables any member of the group - a leaf of the tree - to regularly update the group secret - the tree root. Our contribution addresses a particular weakness of this protocol, clearly identified in the last draft ([BBM+20]): when a user updates the root secret,

[3] In the original working group were Richard Barnes from Cisco, Jon Millican from Facebook, Emad Omara from Google, Katriel Cohn-Gordon both from University of Oxford and Facebook and Raphael Robert from Wire.

he sends different update information to different nodes in the tree. These node secrets shall all be derived from a single secret. Yet, receiving participants can only check the validity of the update information once they have received it, which is too late: if a malicious user sends invalid updates to some part of the tree, then the corresponding users will notice it after they processed it. And the other users that received the correct update information will have moved forward to the new root secret. We introduce a server checking step in the MLS protocol: the goal is that the server shall be able to check whether the update messages are correct, without getting any information about the update secret. The server can then decide to forward or not the update based on the verification of the proof. Our solution is settled on two cryptographic primitives. The first one is a Zero-Knowledge proof, where the public statement is the public key of the node, computed from the new node secret. The second is verifiable encryption, that enables the updater to prove to the server that the update values encrypted for the rest of the tree are the one considered in the Zero-Knowledge proof he supplies.

The public statement (the public key) for the Zero-Knowledge proof is obtained from the secret through a hash like computation (more precisely a HKDF derivation) followed by an algebraic multiplication. Despite the interest of the community in Zero-Knowledge, few propositions exist for Zero-Knowledge proofs on statements that mix algebraic and circuit computations. As an inner contribution, we propose two different protocols to provide a Zero-Knowledge proof for a circuit, given only Pedersen commitments of the input and output. The first is an extension of the recent work of Backes *et al.* ([BHH+19]), based on the MPC in the head paradigm. The second is specific to the case of the circuit representing a pseudorandom function (PRF), and requires some specific properties of a PRF.

1.3 Organisation of this manuscript

The rest of the manuscript is organised as follows: chapter 2 introduces the notations and recalls the definitions of some cryptographic primitives that are requested afterwards. The first part of chapter 3 is dedicated to advent and the precise description of the current Double Ratchet and X3DH algorithms. We also recall the main security analysis given for those in-use protocols. In a second part of this chapter, we explore a more academic side of Ratcheted Key Exchange and investigate on five articles that were published within the five last years. Our chapter 4 details our contributions concerning the multi-device version of the Signal protocol, while chapter 5 is dedicated to our contributions that concern the MLS group messaging protocol, including the Zero-Knowledge part. Finally chapter 6 presents a conclusion of this manuscript and indicates some problems that appeared to us, either as interesting follow up of our contributions, or more generally, as pertinent challenges for the cryptographic community.

Personnal Contributions

[CDDF19] Sébastien Campion, Julien Devigne, Céline Duguey, and Pierre-Alain Fouque. *Multi-Device for Signal*. Cryptology ePrint Archive, Report 2019/1363. https://eprint.iacr.org/2019/1363. 2019 (cit. on p. 83).

[CDDF20] Sébastien Campion, Julien Devigne, Céline Duguey, and Pierre-Alain Fouque. *Multi-Device for Signal*. In: *ACNS 20, Part II*. Ed. by Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi. Vol. 12147. LNCS. Springer, Heidelberg, Oct. 2020, pp. 167–187 (cit. on pp. vii, 7, 83).

[DDF21] Julien Devigne, Céline Duguey, and Pierre-Alain Fouque. *MLS Group Messaging: How Zero-Knowledge Can Secure Updates*. In: *Computer Security – ESORICS 2021*. Ed. by Elisa Bertino, Haya Shulman, and Michael Waidner. Cham: Springer International Publishing, 2021, pp. 587–607 (cit. on pp. 7, 125).

Notations, Definitions and Preliminaries 2

In this chapter, we detail the notations and definitions that will be used in the rest of this document. We start with standard mathematical notations, then continue with the definitions of some cryptographic primitives that appear in our contribution. For each of them, we expose the associated security requirements and elaborate on the related literature if necessary.

Contents

2.1 Mathematical Notations

Sets. We denote by $\mathbb{N}$ the set of natural numbers, $\mathbb{Z}$ the set of integers and $\mathbb{R}$ the real numbers. Let z be an integer, we denote $z \bmod q$ the rest in the euclidean division of z by q. For any $n \in \mathbb{N}$, $\mathbb{Z}_n$ denotes the ring of integers modulo n, $\mathbb{Z}/n\mathbb{Z}$ ($\mathbb{Z}_0 \cong \mathbb{Z}$). For a prime $q \in \mathbb{N}$, $\mathbb{F}_q$ denotes the field of integers modulo q.

Definition 2.1 (Negligible.). *A function* $\mathsf{negl} : \mathbb{N} \to \mathbb{R}^+$ *is said to be negligible if, for every* $c \in \mathbb{N}$, *there exists an integer* n_c *such that, for all* $n \geq n_c$, $\mathsf{negl}(n) \leq n^{-c}$.

Probability. Given a sample space Ω, we denote with $\Pr[e]$ the probability of the specific event e, subset of Ω. The description of the probabilities of the events of Ω is called a distribution of probabilities $\mathcal{D}$ over Ω. We denote by $x \xleftarrow{\mathcal{D}} \Omega$ when x is sampled in Ω following the distribution $\mathcal{D}$. When the sample space is unambiguous, we denote this sampling simply $x \leftarrow \mathcal{D}$.

Uniform distribution. Considering a finite set S, the uniform distribution over S describes the fact that each event in S is observed with equal probability. We note $s \leftarrow\!\!\$\, S$ when s is sampled uniformly (following the uniform distribution) in S.

Definition 2.2 (Indistinguishability of distributions). *Let* $\{\mathcal{D}_n^0\}_{n\in\mathbb{N}}$ *and* $\{\mathcal{D}_n^1\}_{n\in\mathbb{N}}$ *be two sequences of distributions, defined over a set* S_n *for each* n. *One says that* $\{\mathcal{D}_n^0\}_{n\in\mathbb{N}}$ *and* $\{\mathcal{D}_n^1\}_{n\in\mathbb{N}}$ *are:*

- *perfectly indistinguishable if for every* n, $\mathcal{D}_n^0 = \mathcal{D}_n^1$;
- *statistically indistinguishable if there exists a negligible function* negl *such that for all* $n \in \mathbb{N}$,

$$\sum_{i \in S_n} |\Pr_{x\leftarrow\mathcal{D}_n^0}[x = i] - \Pr_{x\leftarrow\mathcal{D}_n^1}[x = i]| \leq \mathsf{negl}((n))\,;$$

- *computationally indistinguishable if for every algorithm* $\mathcal{A}$ *running in poly-time from* $\{0,1\}^n$ *to* $\{0,1\}$, *there is a negligible function* negl *so that for all* $n \in \mathbb{N}$,

$$|\Pr_{x\leftarrow\mathcal{D}_n^0}[\mathcal{A}(x) = 1] - \Pr_{x\leftarrow\mathcal{D}_n^1}[\mathcal{A}(x) = 1]| \leq \mathsf{negl}((n))\,.$$

2.2 Provable Security

In the early stages of cryptography, the difficulty of recovering the information hidden in an encrypted message lied in finding which transformation had undergone the original message. Had the letters been switched? Had the original message been mixed with another? The secret lied more in the treatment - the algorithm - than in the key element (the switch coefficient for instance). In 1883, Auguste Kerchoffs, a linguist from Netherlands, set out, in an article entitled "La cryptographie militaire", six rules that a cryptographic system should verify. Among them, the most famous states that a cryptographic system shall not require secret ("*Il faut qu'il n'exige pas le secret et qu'il puisse sans inconvénient tomber aux mains de l'ennemi.*"). Following this rules, the security must lie in a secret element - the key - that shall be easily communicated. Two questions naturally arise: which tool can be used to design such a cryptographic solution? And how to be sure that a solution is actually secure enough?

A mathematical answer to the first question appears as one-way functions. A function is said to be one-way if it is easy to compute but difficult to invert. Here "easy" shall be understood as

possible in polynomial time while "difficult" means that the probability of success in polynomial time is negligible. For encryption schemes, an extra property is needed, called a trapdoor. A function is said to be a trapdoor one-way function if it is easy to compute, hard to invert, but there exists an additional information: the trapdoor t, such that the inversion of f given t is easy. One-way functions formalize the existence of hard problems, on which are based the modern cryptographic primitives.

For the second question regarding the security, this is precisely the goal of provable security, to provide proofs that a cryptographic function or a protocol, reaches a certain level of security. We first recall some ideas and definitions about hard problems commonly invoked in cryptography, before looking deeper into the domain of game based security proofs. (The habilitation thesis of David Pointcheval [Poi02] was a great help to better understand these notions).

2.2.1 Complexity

In the following, we give some informal definitions concerning the theory of complexity, trying to give an intuition of expressions that will appear in this document, notably the notion of polynomial time.

Turing machines. A Turing machine is a mathematical abstract model of computation. Roughly, it works with a infinite tape on which symbols are written, and on which the machine can perform read and write operations. The machine also keeps in memory its state, that may evolve with each operation. A table determines, for each couple (state of the machine, symbol) the symbol to write and the next move to do: this represents the algorithm. Such a Turing machine is called deterministic. If several possible instructions correspond to each combination (state, symbol), then the machine is said to be non deterministic. In this case, the machine is sometimes called a lucky one because it is supposed to always guess the good combination of instructions to reach the desired result when it is possible. In the following, a Turing machine, if not explicitly described as non deterministic, is supposed deterministic. The running time of a Turing machine corresponds to the number of instructions needed before the machine stops (reaches an ending symbol). In addition, a Turing machine (deterministic or not) can sometimes benefit from an extra random tape, where only random bits are written. In this case, the machine is said to be probabilistic and its running time is more involved, as it depends on the probability of acceptance taken over all the possible random tapes. In this thesis, we will most of the time talk about deterministic or probabilistic algorithms, without the formalism of Turing machines.

NP problems. In complexity theory, problems are ordered in classes, each class gathering problems that are equally difficult to solve, considering a specific factor (time, space, ...). In cryptography, one often considers time complexity. The class P gathers problems that can be decided by a deterministic Turing machine working in polynomial time. That is, the time needed for the machine to end can be written as polynomial in the length of the entry (the symbols written on the tape at the beginning). In terms of complexity theory, this means that the problem can be easily solved. The term easily is to be considered cautiously here. In fact, depending on the polynomial, a problem in the class P can still be difficult to solve in practise. The class NP gathers problems for which a solution can be verified in polynomial time by a deterministic machine but that can only be decided in polynomial time by a non deterministic Turing machine (NP stands for Non deterministic Polynomial time), which means the problem is difficult. A problem is said to be NP-complete if it is in NP and it is at least as difficult as all problems in NP (NP-hard). Hence, NP-complete problems

should represent a great opportunity to design one way functions. However, the difficulty of most NP-complete problems lies on the difficulty of some particular instances and thus nothing insures that any random instance will provide the desired security. The factorization and the discrete logarithm problem, two of the main cryptographic assumptions in use nowadays, are NP problems but not proven in $npol \setminus \mathsf{P}$ neither NP-complete (and highly suspected not to be for the last part).

2.2.2 Game-based proofs

Security parameter. The security of a practical cryptographic algorithm is not something absolute (whereas the ideal one-time pad offers a perfect security). It depends notably on the computational power of an attacker. Considering an unlimited attacker, no primitive can be proven secure. The security parameter, denoted $\lambda \in \mathbb{N}$, can be seen as a bound on the adversarial power. Consequently, the security parameter will also determine the size of the parameters of the algorithm. Considering the complexity theory, the time complexity of the algorithms shall be determined relatively to the length of the input. For key generation algorithms for instance, that shall return a key of length λ having as only input λ, it is necessary to give as an input a unary representation of the security parameter: 1^λ, to have the actual bit size of the security parameter (and not its $\log_2$ value) influence the time complexity.

Adversary. A cryptographic security bound is always defined relatively to an adversary $\mathcal{A}$. The adversary is seen as a probabilistic Turing machine running in time polynomial in the security parameter. We will talk about a PPT adversary.

Security goals. Before proving anything, the first question to answer is: what do I want to prove *i.e.* what is the target of the adversary? The answer depends on the scheme. For instance, secret key encryption schemes may want to protect themselves from an attacker who recovers the secret key. But this is a very difficult goal and proving the security of a scheme for this kind of attacker does not mean the scheme is secure against attacker with smaller ambitions, such as obtaining some information about a plaintext from a given ciphertext. Hence, different degrees of security have been defined for encryption, as we detail in subsection 2.3.1. One can identify two main categories of security goals: computing ones, where the adversary is asked to produce a value, and distinguishing ones, where the adversary is asked to make a difference between two values, usually a value computed as required by the scheme and a random one.

Security experiment. A security experiment is a mental game between the adversary $\mathcal{A}$ and a challenger. The challenger emulates the cryptographic algorithm that is being evaluated. The adversary communicates with the challenger through oracles. The oracles can represent some of the algorithm steps. They can also represent some additional power of the adversary. For instance, the ability to access some side data. To design a security experiment, one has to determine the security goals and the adversary's power.

Adversary's advantage. The security of a scheme ALG considers the probability of success - the advantage - of an adversary $\mathcal{A}$ who plays a security experiment EXP. We denote $\mathsf{Adv}^{\mathsf{EXP}}_{\mathcal{A},\mathsf{ALG}}(\lambda)$ this advantage. In a distinguishing game for instance, a random answer has a chance over two to be correct. The advantage of the adversary is determined as its probability to win compared to the lucky one-half probability. Suppose, without loss of generality, a security experiment EXP where

the challenger samples a bit $b \leftarrow_{\$} \{0,1\}$. If $b = 0$, he returns a value correctly computed by the algorithm ALG that is evaluated. If $b = 1$, he returns a random value with the same characteristics (*e.g.* bit length). The adversary $\mathcal{A}$ returns 1 or 0 depending on which value he thinks was given to him. There are two equivalent ways to measure his advantage. Either one considers his probability of winning, then:

$$\mathsf{Adv}^{\mathsf{EXP}}_{\mathcal{A},\mathsf{ALG}}(\lambda) = \left| \Pr[\mathcal{A} \text{ wins } \mathsf{EXP}] - \frac{1}{2} \right|.$$

Or one measures the probability that $\mathcal{A}$ behaves the same, no matter what value is given to him:

$$\mathsf{Adv}^{\mathsf{EXP}}_{\mathcal{A},\mathsf{ALG}}(\lambda) = \left| \Pr[1 \leftarrow \mathcal{A} | b = 0] - \Pr[1 \leftarrow \mathcal{A} | b = 1] \right|.$$

Both solutions are equivalent up to a factor 2. The second often makes it easier to formalize the security reduction. The final advantage for the security experiment is the maximum of the adversary's advantage, taken over all possible adversaries in the class of the one considered in the experiment (for instance PPT adversaries).

$$\mathsf{Adv}^{\mathsf{EXP}}_{\mathsf{ALG}}(\lambda) = \max_{\mathcal{A}} \mathsf{Adv}^{\mathsf{EXP}}_{\mathcal{A},\mathsf{ALG}}(\lambda).$$

Security reductions. The main goal of a security proof is to show that winning the security experiment is as hard as solving a problem known to be difficult, called a security assumption. Security reductions are an efficient tool inherited from the theory of complexity: given an adversary $\mathcal{A}$ who wins the security experiment, one constructs an adversary $\mathcal{B}$ who calls $\mathcal{A}$ to break the security assumption. The running time t' of $\mathcal{B}$ can be measured as the running time t of $\mathcal{A}$ plus the auxiliary operations required by the simulation. The running time of $\mathcal{B}$ should remain polynomial. As the security assumption is considered as unbreakable in polynomial time (up to some negligible probability), an efficient adversary $\mathcal{A}$ can exist at most with this negligible probability. This is expressed as:

$$\mathsf{Adv}^{\mathsf{EXP}}_{\mathcal{A},\mathsf{Primitive}}(\lambda) \leq L \cdot \mathsf{Adv}^{\mathsf{Sec\ assumption}}_{\mathcal{B}}(\lambda),$$

where L represents a loss factor that can arise in the reduction. In order to use $\mathcal{A}$, $\mathcal{B}$ shall simulate the challenger of $\mathcal{A}$. A particular attention must be taken to the correctness of this simulation, for $\mathcal{A}$ shall not detect that $\mathcal{B}$ simulates his challenger, otherwise $\mathcal{A}$ could chose not to follow his experiment's rules.

Unfortunately, it is rarely possible to provide a direct reduction from the original algorithm to a single security assumption. A game based proof consists in modifying step by step the original game, to end with a game that measures exactly the probability of the adversary to break the security assumption. Each hop from one intermediate game to another should be invisible to the adversary, up to a negligible loss factor. There must be a negligible chance that $\mathcal{A}$ sees that the challenger does not behave as expected in the original game. The final advantage of the adversary is upper bounded by the addition of all those intermediate loss factors, using a triangular inequality.

Asymptotic or concrete security Originally, security reductions did not necessarily compute accurately the reduction loss. In the asymptotic paradigm, as in complexity theory, any polynomial time algorithm is considered as efficient. Consequently, any problem that cannot be decided in polynomial time is difficult. Hence, the only requirement to prove the security is to exhibit a polynomial reduction in the security parameter *i.e.* to build an adversary $\mathcal{B}$ as before with a running time t' "polynomial in the security parameter λ". Given λ is big enough, there exists no efficient

adversary breaking Primitive (as the advantage of $\mathcal{B}$ is negligible in λ). However, this does not give any clue on how to concretely chose λ. That is why concrete security exhorts cryptographers to explicitly express the reduction coefficient hidden in L as well as the running time of $\mathcal{B}$ according to the running time of $\mathcal{A}$ (see for instance [BKR94] or [BR96]). More precisely, a reduction in the concrete security paradigm shall state that, given an adversary $\mathcal{A}$ running in time $t(\lambda)$ who wins the security experiment with probability $\epsilon(\lambda)$, then there exists an adversary $\mathcal{B}$ running in time $t'(\lambda) = T(t, \lambda)$ and breaking the security assumption with probability $E(\epsilon(\lambda), \lambda)$. considering the worst case on each side, this is often written as:

$$\mathsf{Adv}^{\mathsf{EXP}}_{\mathsf{Primitive}}(t) \leq E \cdot \mathsf{Adv}^{\mathsf{Sec\ assumption}}(t'),$$

where t' and E's expressions are fully specified. In order to unify notation throughout the document, we will keep the asymptotic notation, considering the time complexity of the adversary is the worst case time complexity of the experiment. When a concrete security reduction gives precise timing comparison, we will write it explicitly.

Security assumptions based on the discrete logarithm. Let $\mathbb{G}$ be a cyclic group of order q (we denote $\mathbb{G}$ as a multiplicative group but the following results naturally adapt to the additive notation). Let g be generator for this group. Given any element $h \in \mathbb{G}$, the discrete logarithm of h, relatively to g, is the unique element $x \in \mathbb{Z}_q$ such that $h = g^x$. We give three common cryptographic problems on such a group. The difficulty of the problem is expressed as the advantage of an adversary $\mathcal{A}$ who tries to solve the problem. In the following, we implicitly consider that the initial setup $(\mathbb{G}, g, q)$ is generated accordingly to the security parameter, such that the bit length of q is λ.

Definition 2.3 (Discrete Logarithm (DL))**.** *Given a security parameter λ, a cyclic group $\mathbb{G}$ of order q, g a generator of $\mathbb{G}$ and $h \in \mathbb{G}$, compute $x \in \mathbb{Z}_q$ such that $h = g^x$. The advantage of a PPT adversary $\mathcal{A}$ is written as follows:*

$$\mathsf{Adv}^{\mathrm{DL}}_{\mathcal{A},\mathbb{G}}(\lambda) = \Pr\left[h \xleftarrow{\$} \mathbb{G}, x \leftarrow \mathcal{A}(\mathbb{G}, g, q, h) : h = g^x\right].$$

Computing the discrete logarithm is said to be hard in $\mathbb{G}$ if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl *such that:*

$$\mathsf{Adv}^{\mathrm{DL}}_{\mathcal{A},\mathbb{G}}(\lambda) \leq \mathsf{negl}(\lambda).$$

The computational Diffie Hellman problem exploits the difficulty of the discrete logarithm. Given two elements g^x, g^y in $\mathbb{G}$, the challenge is to compute the double exponentiation g^{xy}.

Definition 2.4 (Computational Diffie Hellman (CDH))**.** *Given a security parameter λ, a cyclic group $\mathbb{G}$ of order q, and g a generator of $\mathbb{G}$, the advantage of a PPT adversary $\mathcal{A}$ for the* CDH *problem is written as follows:* $\mathsf{Adv}^{\mathrm{CDH}}_{\mathcal{A},\mathbb{G}}(\lambda) = \Pr[x, y \leftarrow_{\$} \mathbb{Z}_q, Z \leftarrow \mathcal{A}(\mathbb{G}, g, q, g^x, g^y) : Z = g^{xy}]$. *The computational Diffie Hellman assumption is said to be hard in $\mathbb{G}$ if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function* negl *such that:*

$$\mathsf{Adv}^{\mathrm{CDH}}_{\mathcal{A},\mathbb{G}}(\lambda) \leq \mathsf{negl}(\lambda).$$

For the Decisional Diffie Hellman assumption, the adversary is not asked to compute a value but to distinguish between a random element and a value g^{xy}, knowing g^x and g^y. This can be formalized as an experiment, given in Figure 2.1.

$Exp^{\mathrm{DDH}}_{\mathcal{A},\mathbb{G}}(\lambda)$

1 : $x, y, z \leftarrow_{\$} \mathbb{Z}_q$
2 : $b \leftarrow_{\$} \{0,1\}$
3 : $X \leftarrow g^x, Y \leftarrow g^y$
4 : $Z_0 \leftarrow g^{xy}$
5 : $Z_1 \leftarrow g^z$
6 : $b' \leftarrow \mathcal{A}(\mathbb{G}, g, X, Y, Z_b)$
7 : **return** b'

Figure 2.1 – The Decisional Diffie-Hellman security experiment.

Definition 2.5 (Decisional Diffie Hellman (DDH))**.** *Given a security parameter λ, a cyclic group $\mathbb{G}$ of order q, and g a generator of $\mathbb{G}$, the advantage of PPT adversary $\mathcal{A}$ is written as follows:*

$$\mathsf{Adv}^{\mathrm{DDH}}_{\mathcal{A},\mathbb{G}}(\lambda) = \left|\Pr\left[1 \leftarrow Exp^{\mathrm{DDH}}_{\mathcal{A},\mathbb{G}}(\lambda) | b = 0\right] - \Pr\left[1 \leftarrow Exp^{\mathrm{DDH}}_{\mathcal{A},\mathbb{G}}(\lambda) | b = 1\right]\right|.$$

The Decisional Diffie Hellman assumption is said to be hard in $\mathbb{G}$ if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl *such that:*

$$\mathsf{Adv}^{\mathrm{DDH}}_{\mathcal{A},\mathbb{G}}(\lambda) \leq \mathsf{negl}(\lambda).$$

These problems are listed here from the most difficult to the easier one. For given $\mathbb{G}$ and g, any instance of the discrete log problem can be reduced to a random instance. Suppose one has to determine $x = \log_g y$. Then one can sample $a \in \mathbb{Z}_q$, set $Y = y^a$. Then if one finds $X = \log_g Y$, $X/a \bmod q$ is a valid value for x. This reduction is only valid if q is prime. There also exists an additive reduction, valid for any q: if one finds $X = \log_g yg^a$ then $x = X - a \bmod q$. However, the multiplicative one also shows that the difficulty is independent of g (x is a valid discrete log of Y relatively to g^a). Finally, if one can solve a non negligible number of instances in polynomial time, then one can solve all instances in average polynomial time. One can expect all the instances to be equivalently hard.

The difficulty of those problems depends on the structure of the group $\mathbb{G}$. If $\mathbb{G}$ has no specific algebraic structure that can help, then only generic algorithms, such as the Pollard-rho method ([Pol78]) are avalaible. These generic algorithms require at least $\sqrt{q}$ operations.

A new problem, called Gap DDH was introduced in [OP01]. The idea is to measure the difficulty to solve the CDH problem given the access to a DDH oracle.

Definition 2.6 (Gap Diffie Hellman (gDDH))**.** *Given a security parameter λ, a cyclic group $\mathbb{G}$ of order q, and g a generator of $\mathbb{G}$, the advantage of a PPT adversary $\mathcal{A}$ for the* CDH *problem is written as follows:*

$$\mathsf{Adv}^{\mathrm{gDDH}}_{\mathcal{A},\mathbb{G}}(\lambda) = \Pr\left[x, y \leftarrow_{\$} \mathbb{Z}_q, Z \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{DDH}}}(\mathbb{G}, g, q, g^x, g^y) : Z = g^{xy}\right],$$

where the oracle $\mathcal{O}_{\mathrm{DDH}}$ is defined as follows: on a query $(g, g^x, g^y, Z) \in \mathbb{G}$, it returns 1 if $Z = g^{xy}$, 0 otherwise. The gap Diffie Hellman assumption is said to be hard in $\mathbb{G}$ if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl *such that:*

$$\mathsf{Adv}^{\mathrm{gDDH}}_{\mathcal{A},\mathbb{G}}(\lambda) \leq \mathsf{negl}(\lambda).$$

Elliptic curves cryptography. A common concrete instantiation of a Diffie Hellman group is based on elliptic curves. An elliptic curve is an algebraic curve on a field $\mathbb{K}$ satisfying an equation of the form:

$$y^2 = x^3 + ax + b, a, b \in \mathbb{K}, (\text{if } car(\mathbb{K}) \neq 2, 3).$$

In cryptography, we consider elliptic curves defined on a finite field $\mathbb{F}_p$, and denote them $E(\mathbb{F}_p)$. A curve $E(\mathbb{F}_p)$, together with a virtual point P_∞, composes an additive group. The addition and doubling laws can easily be determined geometrically, we do not detail them here. The number of points in $E(\mathbb{F}_p)$ is bounded by Hasse's theorem and can be considered close to the number of elements in the field of definition.

$$|\sharp E(\mathbb{F}_p) - (p-1)| \leq 2\sqrt{p}.$$

Usually, one look for elliptic curves on a prime order field $\mathbb{F}_p$, with a point $P \in E(\mathbb{F}_p)$ of order q such that q is close to p. More precisely, q divides $\sharp E(\mathbb{F}_p)$ such that $\sharp E(\mathbb{F}_p) = mq$ with m small. Then $\langle P \rangle$ is a cyclic group on which the discrete logarithm is hard. Considering a generic curve, the most efficient algorithms computes a discrete logarithm in $\mathcal{O}\left(\sqrt{q}\right)$ Hence one has to choose a curve with order twice longer than the security parameter. However, one has to be cautious on the choice of the curve because some specific curves have more efficient algorithms.

2.3 Basic cryptographic primitives

2.3.1 Encryption

Encryption is maybe the most famous and the most ancient cryptographic primitive. Introduction courses or popularization books often begin with old classics as the Caesar cipher. This basic scheme consists in replacing each letter of the original by the letter situated some fixed position after in the alphabet. Knowing the shift rate, the receiver processes the inverse shift and recovers the message. This toy example is indeed a good introduction to symmetric encryption. Symmetric states that the sender and the receiver play symmetric roles, as they both use a common secret key (in the Caesar cipher, the shift rate) to encrypt and decrypt the message.

Definition 2.7 (Symmetric encryption)**.** *A symmetric encryption scheme* SE*, with keyspace $\mathcal{K}$, message space $\mathcal{M}$, and cipher space $\mathcal{C}$, is composed of three algorithms:*

- $\mathsf{KeyGen} : 1^\lambda \to \mathcal{K}$ *which on input a security parameter λ (given in unary representation), outputs a secret key* k*;*
- $\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$*, a probabilistic algorithm which, on input a secret key* k *and a message m, outputs a ciphertext c;*
- $\mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M} \times \perp$ *which on input a secret key* k *and a ciphertext c, outputs a message m or $\perp$ if the decryption was unsuccessful.*

Sometimes KeyGen is preceded with a fourth algorithm SetUp that, on input the security parameter, returns common set up information. As it is most of the time omitted, we do not include it in the definition. Considering Enc is a randomized algorithm, it may sometimes be of interest to make the randomness visible. In those cases, we will write $\mathsf{Enc}(\mathsf{k}, m : r)$ for the encryption of a message m under the key k using randomness r.

The main difficulty of symmetric encryption is the sharing of a secret key. When the sender and the receiver can not rely on a common value, they can use a public key encryption scheme. In this setting, the sender hides the message using a public key, which is publicly available, as its name suggests. The recipient can decrypt using a private key which he is the only one to know.

Definition 2.8 (Public Key Encryption)**.** *A public key encryption scheme* PKE *with secret key space $\mathcal{SK}$, public key space $\mathcal{PK}$, message space $\mathcal{M}$, and cipher space $\mathcal{C}$, is composed of the three following algorithms:*

- $\mathsf{KeyGen} : 1^\lambda \to \mathcal{SK} \times \mathcal{PK}$*, which, on input a security parameter, outputs a key pair composed of a private key* sk *and a public key* pk*;*
- $\mathsf{Enc} : \mathcal{PK} \times \mathcal{M} \to \mathcal{C}$*, a probabilistic algorithm which, on input a public key* pk *and a message m, outputs a ciphertext c;*
- $\mathsf{Dec} : \mathcal{SK} \times \mathcal{C} \to \mathcal{M} \times \perp$*, a deterministic algorithm which, on input a private key* sk *and a ciphertext c, outputs a message m or $\perp$ if the decryption was unsuccessful .*

Security definitions. In both cases, symmetric or public key encryption, the purpose is to protect the content of the message (also called the plaintext) from an eavesdropper who would not know the secret/private key, while keeping a practical algorithm. In order to prove that a primitive satisfies this security goal, one needs a precise definition of what protecting means. The basic property one expects from an encryption scheme is that, when following the protocol with the correct keys, then a ciphertext decrypts to the original plaintext. This property is called correctness.

Definition 2.9 (Correctness)**.** *A public key encryption scheme (respectively a symmetric encryption scheme) is said to be correct if, for all key pairs* $(\mathsf{sk}, \mathsf{pk}) \in \mathcal{SK}$ *(resp. for all key* $\mathsf{k} \in \mathcal{K}$*), for all messages $m \in \mathcal{M}$,* $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(pk, m)) = m$ *(resp.* $\mathsf{Dec}(\mathsf{k}, \mathsf{Enc}(\mathsf{k}, m)) = m$*).*

The notion of semantic security was introduced by Goldwasser and Micali in [GM82]. Informally, this property states that "an adversary, who knows the encryption algorithm and is given the ciphertext, cannot obtain any information about the cleartext". Another definition, proven equivalent ([GM84]), is based on the notion of indistinguishability. This notion is the one that is commonly used to prove the security of encryption scheme, as it is easier to express and understand. The idea is that the encryption scheme reaches its goal if an adversary has negligible chances to distinguish between the encryption of two different plaintext that he knows. If the adversary is only given access to the public key (in the symmetric setting he is given an encryption oracle), then it corresponds to *indistinguishability under chosen plaintext attack* (IND-CPA security). If he is also given access to a decryption oracle, then the corresponding property is called *indistinguishability under chosen ciphertext attack* (IND-CCA security).

We detail in Figure 2.2 the traditional IND-CPA and IND-CCA security experiments in the asymmetric setting. Equivalent definitions exist for the symmetric setting. In this case, the adversary is given access to an encryption oracle in place of the public key.

We give in Figure 2.3 another indistinguishability experiment, in which the adversary is asked to distinguish between the encryption of a plaintext he knows and the encryption of a random plaintext of the same size. We identify these variations as ror-CPA and ror-CCA because they are related to the real-or-random (RoR) definition of indistinguishability (for both CPA and CCA case) defined in [BDJR97] in the symmetric context, the main difference being that, in the latter,

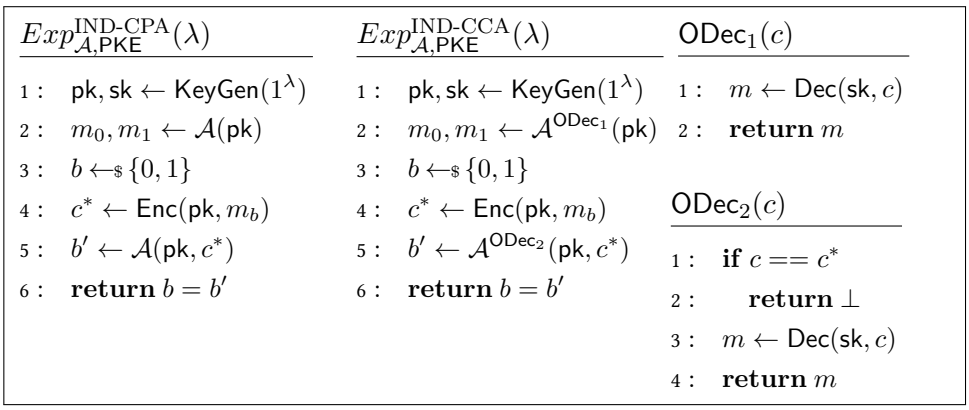


Figure 2.2 – IND-CPA and IND-CCA security experiments.

the adversary is given access to a real-or-random oracle instead of submitting a single plaintext. The same way, the authors define a left-or-right (LoR) indistinguishability experiment (again in the CPA and CCA version), which is similar to the IND-CPA and IND-CCA experiments described in Figure 2.2, but in the symmetric case and with the access to a left-or-right oracle instead of a single pair of message to be submitted.

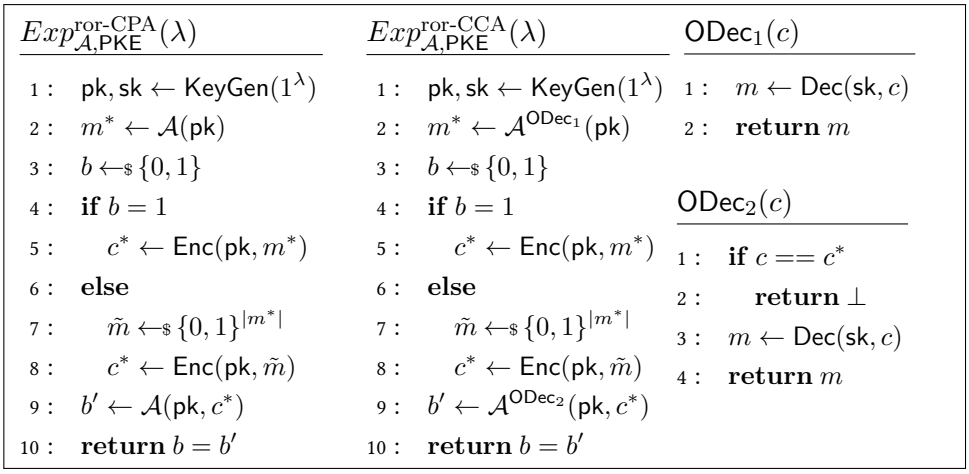


Figure 2.3 – ror-CPA and ror-CCA security experiments.

For concision, we denote with VRS the version of the experiment we consider - IND or ror - and with ATK the kind of attack, CPA or CCA. With these notations in mind, we can write the following definition:

Definition 2.10 (IND-CPA/IND-CCA/ror-CPA/ror-CCA security)**.** *For* $\text{VRS} \in \{\text{IND}, \text{ror}\}$ *and* $\text{ATK} \in \{\text{CPA}, \text{CCA}\}$*, a public key encryption scheme* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be* VRS-ATK *secure if, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that:*

$$\mathsf{Adv}^{\text{VRS-ATK}}_{\mathcal{A},\mathsf{PKE}}(\lambda) = \left| \Pr\left[\mathsf{Exp}^{\text{VRS-ATK}}_{\mathcal{A},\mathsf{PKE}}(\lambda) = 1\right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)\,.$$

And $\mathsf{Adv}^{\text{VRS-ATK}}_{\mathsf{PKE}}(\lambda) = \max_{\mathcal{A}}(\mathsf{Adv}^{\text{VRS-ATK}}_{\mathcal{A},\mathsf{PKE}}(\lambda))$*.*

Note that for a symmetric encryption scheme, the number of queries to the encryption oracle may be bounded by some $q_e \in \mathbb{N}$. This is necessary to formalise a concrete reduction and to turn the advantage into practical recommendations. For instance, q_e will bound the number of encryption that can be done with a single key.

The authors of [BDJR97] give reductions between their LoR and RoR versions of indistinguishably, that are transposable in the public key setting (mentioned in [HMS03] for instance) and the single message case. We give the reduction from IND-CPA to ror-CPA. The CCA version follows easily and is the one that we will call in chapter 4.

Proposition 2.1 (IND-CPA implies ror-CPA.)**.** *For any public key encryption scheme* PKE *as defined in definition 2.8,*

$$\mathsf{Adv}_{\mathsf{PKE}}^{\text{ror-CPA}}(\lambda) \leq \mathsf{Adv}_{\mathsf{PKE}}^{\text{IND-CPA}}(\lambda).$$

Proof. Let $\mathcal{A}_1$ be an adversary attacking a scheme PKE following the ror-CPA experiment. $\mathcal{A}_1$ We present a second adversary $\mathcal{A}_2$ that uses $\mathcal{A}_1$ to attack PKE in the IND-CPA sense. $\mathcal{A}_2$ simulates the role of the ror-CPA challenger for $\mathcal{A}_1$ and answers his queries as follows:

1. At the beginning of the experiment, $\mathcal{A}_2$ receives a public key pk and transmits it to $\mathcal{A}_1$.
2. When receiving the plaintext m^* sent by $\mathcal{A}_1$, $\mathcal{A}_2$ identify it as m_1, sample a second plaintext m_0 of the same size and transmits the pair (m_0, m^*) to its own challenger.
3. When receiving an encryption c_b, that depends on the bit b sampled by his challenger, $\mathcal{A}_2$ transmits it to $\mathcal{A}_1$.
4. Following the experiment, $\mathcal{A}_1$ answers to $\mathcal{A}_2$ with a bit b' and $\mathcal{A}_2$ answer to his challenger with the same bit b'.

We write the advantage of $\mathcal{A}_2$ as:

$$\mathsf{Adv}_{\mathsf{PKE}}^{\mathcal{A}_2,\text{IND-CPA}}(\lambda) = \left|\Pr[1 \leftarrow \mathcal{A}_2 | b = 1] - \Pr[1 \leftarrow \mathcal{A}_2 | b = 0]\right|.$$

which is, as recalled in section 2.2.2, equivalent to the expression given in definition 2.10.

If $b = 1$, then the game played by $\mathcal{A}_1$ is exactly the experiment ror-CPA with $b = 1$. Hence $\Pr[1 \leftarrow \mathcal{A}_2 | b = 1] = \Pr[1 \leftarrow \mathcal{A}_1 | b = 1]$.

If $b = 0$, then the game played by $\mathcal{A}_1$ is exactly the experiment ror-CPA with $b = 0$.

Finally,

$$\mathsf{Adv}_{\mathsf{PKE}}^{\mathcal{A}_2,\text{IND-CPA}}(\lambda) = \left|\Pr[1 \leftarrow \mathcal{A}_1 | b = 1] - \Pr[1 \leftarrow \mathcal{A}_1 | b = 0]\right| = \mathsf{Adv}_{\mathsf{PKE}}^{\mathcal{A}_1,\text{ror-CPA}}(\lambda).$$

As this is true for any adversary $\mathcal{A}_1$ against ror-CPA, one gets $\mathsf{Adv}_{\mathsf{PKE}}^{\text{ror-CPA}}(\lambda) \leq \mathsf{Adv}_{\mathsf{PKE}}^{\text{IND-CPA}}(\lambda)$. □

The IND-CCA to ror-CCA is obtained exactly the same way, as $\mathcal{A}_2$ can perfectly answer to the decryption queries of $\mathcal{A}_1$ calling its own decryption oracle.

About the IND-CCA *definition.* Our definition of IND-CCA security is sometimes referred to as IND-CCA2 security, or adaptive IND-CCA. The IND-CCA1 case, considered in [NY90] for instance, corresponds to the experiment where the adversary is given access to the decryption oracle before he sends his challenge messages m_0, m_1 but no longer after he has received the challenged ciphertext c^*[1].

[1] It is sometimes identified as the lunch-time attack as to a real adversary that would access the target computer during the owner's lunch time.

A classical public key example : ElGamal encryption scheme. We detail hereafter the ElGamal encryption scheme, as an example of an IND-CPA secure construction. We present it in the traditional multiplicative group version but it transposes to an additive group (elliptic curves). Consider a cyclic group $\mathbb{G}$ of prime order q, and g a generator for this group. Let $\mathcal{M} = \mathbb{G}$ be the message space, $\mathcal{PK} = \mathbb{G}^*$ be the public key space and $\mathcal{SK} = \mathbb{Z}_q^*$ be the private key space. The KeyGen algorithm consists in sampling a private key sk in $\mathbb{Z}_q^*$ and to define the corresponding public key as $\mathsf{pk} = g^{\mathsf{sk}}$. Then the encryption of a group element m is defined as : $\mathsf{Enc}(pk, m : r) = (g^r, pk^r \cdot m) = (c_1, c_2)$ where r is a random element in $\mathbb{Z}_q^*$. The decryption algorithm proceeds as follow : $\mathsf{Dec}(\mathsf{sk}, (c_1, c_2)) = c_2/c_1^{\mathsf{sk}}$. Considering that $\mathbb{G}$ is a group where the Decisional Diffie Hellman problem is hard, ElGamal is an IND-CPA public key encryption scheme. Some more example that we do not detail: RSA-OAEP is proven IND-CCA secure in [FOPS04]. In [CS98], Cramer and Shoup also propose an IND-CCA secure scheme, the first to be proven secure without using a random oracle[2]. Another IND-CCA secure solution based on Elliptic Curve (known as $ECIES$) is defined in [Sho01].

Public Key Infrastructure. When one encrypts a message using a public key scheme, one has to be sure that the public key he uses indeed belongs to the intended recipient. A common solution is to have a third party - the public key infrastructure (PKI) - binds the public key to the owner of the corresponding private key. This is often realised through a certificate authority, that delivers a certificate for the public key.

The hybrid encryption or the KEM-DEM paradigm. As we have seen earlier, the traditional asymmetric encryption solved the major problem of allowing encryption when no secret key is shared among the users. However, this is done with an non negligible cost on efficiency. Public key encryption schemes rely on mathematically based cryptographic assumptions such as big number factorisation (RSA [RSA78]) or discrete logarithm computation. Consequently, encryption and decryption mechanism require to perform some heavy computation compared to classical symmetric schemes. Hybrid encryption naturally appeared as a way to take the best of both worlds. The idea is to encrypt the smallest amount of information: a key, using a public key encryption scheme, then to encrypt the data with a symmetric scheme, so as to save time and bandwidth. This is often identified as the KEM-DEM (Key Encapsulation Mechanism - Data Encryption Mechanism) paradigm, as formalised in [Sho01], where KEM part consists in the asymmetric encryption of the symmetric key.

2.3.2 Multi-user, multi-recipient and broadcast encryption

In the real world, an adversary's knowledge will not be limited to seeing communications corresponding to a single (public key, private key) pair. It is commonly needed to encrypt a same content to different receivers. This is called broadcast or multicast. It is frequently used for copyright management for instance, or for on demand/registration based video services. At a network level, broadcast and multicast have a slightly different signification. A broadcast consists, for a user, to send a packet to all the users of a network, using the specific broadcast IP address for the network for instance. The sender may or not be itself a member of the network. A multicast does not target a whole network, it selects the users concerned with the packet. Moreover, it is not restricted to a single network: nodes from different networks can be gathered under a same multicast IP address.

[2]The random oracle model is detailed in subsection 2.5.1.

However at a cryptographic level, broadcast, multicast or even multi-recipient encryption often mingle a little.

Encryption in a multi-user setting A naive solution to encrypt a unique content to n users that do not share a common key, is to provide each user with a personal (private, public) key pair. Then anyone can encrypt a message to all users or to any subgroup of t users. In this precise case, the adversary gets an easy access to a single message encrypted under several public keys, which is out of the traditional frame of IND-CPA and IND-CCA definitions. Baudron *et al.* address this problem, called multi-user setting, in [BPS00]. They even consider a broader problem, that consists in an adversary who sees encryption of related messages under several keys. The authors show an equivalence between the single user security - the traditional IND-CPA or IND-CCA - and the multi user security, as long as the number of users, n, remains polynomial. The reduction from the multi-user case to the single user case grows the advantage of the adversary by a factor n. The security model follows the single user one except that the adversary is now expected to provide two vectors of plaintext of dimension n, only one of which (depending on the challenger bit b) will be encrypted and returned.

In [BBM00], Bellare *et al.* address the same problem. They also show that both the IND-CPA and IND-CCA properties can be translated from the single user to the multi-user setting. However, their model is slightly different. Instead of providing two vectors of plaintext, their adversary is given access to several oracles. Each public key pk_i corresponds to a left or right oracle OLR_i that, on input two messages of equal length m_0, m_1 and a test bit b, returns the encryption of m_b under the public key pk_i. The test bit b sampled by the challenger, who instantiates these oracles, is unique. We detail in Figure 2.4 the security experiment for n user IND-CCA security. A similar experiment without decryption oracles corresponds to n-CPA security. We recall in Theorem 2.2 the general reduction result of [BBM00]. This possibility of submitting several pairs of messages to the challenger captures the ability for the adversary to see encryptions of related plaintext under different keys in an adaptive way. The result is similar to [BPS00], except for the loss factor which is augmented due to the adaptiveness. The general security reduction induces a loss factor in nq_e where n is the number of users and q_e is the number of queries to the OLR oracles authorized in the experiment.

$Exp^{\text{n-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda)$

1 : **for** $i = 1 \dots n$
2 : $\quad \mathsf{pk}_i, \mathsf{sk}_i \leftarrow \mathsf{KeyGen}(1^\lambda)$
3 : $b \leftarrow\!\!\$ \{0,1\}$
4 : $b' \leftarrow \mathcal{A}^{\mathsf{OLR}_1^b, \dots, \mathsf{OLR}_n^b, \mathsf{ODec}_1, \dots \mathsf{ODec}_n}(pk_1, \dots, pk_n)$
5 : **return** $b = b'$

$\mathsf{ODec}_i(c)$

1 : **if** $c = c_i^*$ **return** $\perp$
2 : $m \leftarrow \mathsf{Dec}(\mathsf{sk}_i, c)$
3 : **return** m

$\mathsf{OLR}_i^b(m_0, m_1)$

1 : $c_i^* \leftarrow \mathsf{Enc}(\mathsf{pk}_i, m_b)$
2 : **return** c_i^*

Figure 2.4 – The n-CCA security experiment.

Definition 2.11 (n-CCA security). *A public key encryption scheme* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be* n-CCA *secure if, for any PPT adversary* $\mathcal{A}$*, making at most* q *queries to the decryption oracle and* q_e *queries to its left-or-right oracles, there exists a negligible function* negl *such that:*

$$\mathsf{Adv}^{\text{n-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\text{n-CCA}}_{\mathcal{A},\mathsf{PKE}}(\lambda) = 1\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)\,.$$

We denote $\mathsf{Adv}^{\text{n-CCA}}_{\mathsf{PKE}}(\lambda, q, q_e)$ *the advantage for this experiment.*

The IND-CCA definition considered in [BBM00] corresponds to IND-CCA2, where the adversary is given access to the decryption oracle even after its challenge queries.

Theorem 2.2 (Multi user public key encryption reduction)**.** *Let* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme. Let* n*,* q*,* q_e *be integers.*

$$\mathsf{Adv}^{\text{n-CCA}}_{\mathsf{PKE}}(\lambda, q, q_e) \leq nq_e\mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{PKE}}(\lambda, q).$$

Where the maximum on the left side is taken over adversaries $\mathcal{A}$ *running in time* t*, and the maximum on the right side is taken over adversaries* $\mathcal{B}$ *running in time* $t' = t + \mathcal{O}(\log(q_e n))$*.*

The security loss in this general reduction is important but is shown to be as tight as possible for a general reduction. However, the authors of [BBM00] also propose reductions for two specific schemes, namely the ElGamal (*cf.* section 2.3.1) and the Cramer-Shoup encryption schemes ([CS98]), both based on the DDH hypothesis, for which they reach better security bounds. We do not detail those results here and refer to the original paper for more details.

Extending the work of Kurosawa [Kur02], the authors of [BBKS07] study another variant, called multi-recipient encryption together with a security model. They especially consider the possibility and the consequences on security of reusing randomness in the context of multi recipient encryption. As they explain, reusing randomness while encrypting two different messages under a single public key can dramatically damage the security. But they show that reusing randomness while encrypting messages to different public key can be done securely, assuming a property - reproducibility - on the base encryption scheme. The reproducibility test for a given encryption scheme Enc, considers the possibility, given a public key pk_1, the encryption c_1 of a random message m_1 under pk_1 with randomness r ($c_1 = \mathsf{Enc}(pk_1, m_1 : r)$, a (public,private) key pair (pk_2, sk_2) and another random message m_2, to produce in polynomial time the encryption of m_2 under pk_2 using the same randomness r. The authors show that classical discrete log based encryption schemes, such as ElGamal ([ElG84], *cf.* section 2.3.1) or DHIES ([ABR01]), which follows the KEM-DEM paradigm, are reproducible, and so extend naturally to a random reuse multi recipient version. This work potentially enables great efficiency savings while implementing broadcast with a naive solution. In particular, considering DHIES, this result offers the possibility to encrypt a single symmetric key (single message multi recipient encryption) with single randomness (randomness reuse) for all the recipients. However, their result is not focused on the single-message setting. Barbosa and Farshim extend this work to the single message setting in [BF07]. They particularly show that a relaxation in the definition of reproducibility can lead to efficient specific single key multi recipient Key encapsulation mechanisms, as introduced in [Sma05], and propose a construction from an adaptation of the Cramer Shoup scheme ([CS98]).

Broadcast Encryption. The main drawback of the naive solution is easily seen : its inefficiency. This solution implies t encryptions of the message, for t recipients, which can be prohibitive when t is large. The global public key of all possible users also grows linearly with the number n of participants. However, on the positive side, each user has a constant secret key which size is independent from the different subsets considered. On the road for better solutions, Fiat and

Naor introduced in [FN94] a dedicated cryptographic primitive, called broadcast encryption. In this seminal work, a broadcast encryption scheme involves a centre that generates keys for the participants and encrypts information to a dynamically changing set of decoders, chosen among a predefined collection of n users. At encryption time, the sender can choose, within the enrolled n enrolled participants, a subset $\mathcal{S}$ actually targeted by the message. Depending on the scheme, this can be done by positively defining the set of receivers or, on the "negative side" by excluding users from the global set of participant. Among the security properties that a broadcast scheme can achieve is collusion resistance: even if the users outside of the targeted set $\mathcal{S}$ collude, they do not obtain any information on the message. Public key versions of broadcasting, introduced in the context of traitor tracing ([TT01], [DF03] for instance), enable any user to encrypt information to a designated group of users, only leaving to the centre the task of performing the set up phase (the generation of the keys) and managing the group. The main differences with multi-recipient encryption are the pre-defined pool of participants and the fact that a centre generates the keys for the participants, which leads to different schemes.

The research in the domain of broadcast consists in proposing secure schemes, with improved properties, while achieving the better trade-off between the ciphertext, the public key and the private keys size. We give hereafter an overview of some important contributions in the domain, that will be useful later on in this manuscript to understand why we did not rely on this primitive in our contribution. We describe the practical aspects of the scheme and do not detail their security properties. In 2011, Hieu Phan *et al.* proposed in [PPS11] an overview of the different security notions for broadcast encryption. We refer to this publication for more detail.

Two efficient collusion resistant schemes were proposed by Boneh, Gentry and Water in [BGW05]. The first with small constant size ciphertext and private decryption keys. As in the naive solution, the public encryption key grows linearly with the number n of participants in the targeted set. The second solution has ciphertext, encryption and decryption keys in $\mathcal{O}(\sqrt{n})$.

In [DPP07], Delarablée *et al.* introduced a dynamic version of broadcast encryption. The given definition of a dynamic scheme especially requires that the ciphertext size shall not depend on the number of participants. In that sense, the naive solution is not dynamic. In this scheme, new users can join the initial pool of potential receivers, with the property that newcomers are able to decrypt previously encrypted messages, and some other can be revoked. The public encryption key size is $\mathcal{O}(n)$, the private decryption key size is constant (the decryption keys are not impacted by the addition of a user) and the ciphertext grows linearly with the number r of participants in the targeted set $\mathcal{S}$.

An efficient scheme, using identity based encryption is described in [Del07]. In this proposition, the total number of participant can be infinite as it corresponds to the number of identities. The security depends on the maximum size of a set of receivers, not in the number of decryption keys. The scheme is not dynamic relatively to the definition of [DPP07]. However any identity can be included in the target set of identities that should receive the encrypted message. Hence, as in the naive solution, there is no need for specific join or revoke algorithms, but this requires an upper bound on the cardinality of the target set. This schemes moreover does not authorize the newcomers to access the previously exchanged messages (this property is called forward secrecy against new users, however we will not use this denomination as it can be confusing). The private decryption key and ciphertext have constant size while the public encryption key grows linearly with the maximum size of the set of receiver for one encryption. All the previous schemes rely on a central administration.

In [PPS12], the authors propose a decentralized instantiation of dynamic broadcast encryption. The centre is eliminated, and replaced by an interactive group key exchange within the participant. Their

construction is dynamic in the sense of [DPP07] except for the modification of private decryption key when adding a new participant. This choice is made on purpose to prevent the newcomer to read past messages. As we see, dynamic broadcast encryption realises multicast communications and, in the cryptographic literature, both terms are sometimes used equally. Most of the broadcast encryption scheme (including those described above) are based on the KEM-DEM paradigm. Logically, the complexity lies in the group key encapsulation part. Hence broadcast encryption is also heavily bound to group key establishment.

2.3.3 Message authentication codes

A message authentication code is a symmetric key primitive. From a message and a symmetric key, it computes a single value - the tag - which shall provide two desirable properties. Firstly it provides information about the sender's identity. Secondly, it guarantees that the message has not been modified along the way between the sender and the receiver. This last feature is known as integrity.

Definition 2.12 (Message Authentication Code (MAC))**.** *A message authentication code* MAC *is composed of three algorithms:*

- $\mathsf{KeyGen}_{\mathtt{m}} : 1^{\lambda} \to \mathcal{K}_{\mathtt{m}}$*, which, on input a security parameter, returns a secret key* $K_{\mathtt{m}}$*;*
- $\mathsf{Mac} : \mathcal{K}_{\mathtt{m}} \times \mathcal{M} \to \mathcal{T}$*, which, on input a key* $K_{\mathtt{m}}$ *and a message* m*, outputs a tag* t*;*
- $\mathsf{Verif} : \mathcal{K}_{\mathtt{m}} \times \mathcal{M} \times \mathcal{T} \to \{\mathsf{true}, \mathsf{false}\}$*, which, on input a secret mac key* $K_{\mathtt{m}}$*, a message* m*, and a tag* t*, returns* true *if the verification succeeds,* false *otherwise.*

The notion that formalizes the security we expect from a MAC scheme is called unforgeability. Namely, one requires that it is almost infeasible for an adversary, who does not know the secret mac key, to compute a tag t corresponding to a message of its choice. It follows that it must be a person who knows the key who computed the tag. And the message attached to the tag can not be modified, otherwise, the adversary would also have to compute the corresponding tag. This is formalized by two experiments, existential unforgeability and strong unforgeability, detailed in Figure 2.5. The first notion excludes the cases where the adversary tries to forge a mac on a message, given a first tag on this same message. If the scheme is deterministic, this attack has no sense. In the case where the MAC algorithm is probabilistic (takes as extra input a random sampled from a random space $\mathcal{R}$), then the strong unforgeability notion excludes the attack.

$Exp^{\mathrm{EUF}}_{\mathcal{A},\mathsf{MAC}}(\lambda)$	$Exp^{\mathrm{SUF}}_{\mathcal{A},\mathsf{MAC}}(\lambda)$	$\mathsf{OMac}(m)$
1 : $K_{\mathtt{m}} \leftarrow \mathsf{KeyGen}_{\mathtt{m}}(1^{\lambda})$	1 : $K_{\mathtt{m}} \leftarrow \mathsf{KeyGen}_{\mathtt{m}}(1^{\lambda})$	1 : $t \leftarrow \mathsf{Mac}(K_{\mathtt{m}}, m)$
2 : $(m^*, t^*) \leftarrow \mathcal{A}^{\mathsf{OMac}}$	2 : $(m^*, t^*) \leftarrow \mathcal{A}^{\mathsf{OMac}}$	2 : $R_1 \leftarrow m$
3 : $v \leftarrow \mathsf{Verif}(K_{\mathtt{m}}, m^*, t^*)$	3 : $v \leftarrow \mathsf{Verif}(K_{\mathtt{m}}, m^*, t^*)$	3 : $R_2 \leftarrow (m, t)$
4 : **if** $m^* \notin R_1 \wedge v == \mathsf{true}$	4 : **if** $(m^*, t^*) \notin R_2 \wedge v == \mathsf{true}$	4 : **return** t
5 : **return** 1	5 : **return** 1	
6 : **else return** 0	6 : **else return** 0	

Figure 2.5 – EUF and SUF security experiments.

Definition 2.13 (EUF security). *A* MAC *scheme* $\mathsf{MAC} = (\mathsf{KeyGen}_{\mathtt{m}}, \mathsf{Mac}, \mathsf{Verif})$ *is said to be* EUF *secure if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that:*

$$\mathsf{Adv}^{\mathrm{EUF}}_{\mathcal{A},\mathsf{MAC}}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\mathrm{EUF}}_{\mathcal{A},\mathsf{MAC}}(\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)\,.$$

We denote $\mathsf{Adv}^{\mathrm{EUF}}_{\mathsf{MAC}}(\lambda)$ *the advantage for this experiment.*

Definition 2.14 (SUF security). *A* MAC *scheme* $\mathsf{MAC} = (\mathsf{KeyGen}_{\mathtt{m}}, \mathsf{Mac}, \mathsf{Verif})$ *is said to be* SUF *secure if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that:*

$$\mathsf{Adv}^{\mathrm{SUF}}_{\mathcal{A},\mathsf{MAC}}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\mathrm{SUF}}_{\mathcal{A},\mathsf{MAC}}(\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)\,.$$

We denote $\mathsf{Adv}^{\mathrm{SUF}}_{\mathsf{MAC}}(\lambda)$ *this advantage.*

In this work, we will need an extended version of strong unforgeability, in a multi-instance setting as defined in [PR18a]. We recall the corresponding experiment in Figure 2.6. The definition is similar to the classical SUF except that the adversary can play with several instances for the MAC, corresponding to different keys, and can expose those keys. He will try to forge a new (message, tag) pair for an instance for which the key has not been exposed. The reduction from the multi instance version to the classical strong unforgeability induces a loss factor in the number of instances.

$Exp^{\text{mi-SUF}}_{\mathcal{A},\mathsf{MAC}}(\lambda)$

1 : $k \leftarrow 0$
2 : $(i, m^*, t^*) \leftarrow \mathcal{A}^{\mathsf{OMac},\mathsf{OGen}}$
3 : Require $1 \leq i \leq k$
4 : $v \leftarrow \mathsf{Verif}(K^i_{\mathtt{m}}, m^*, t^*)$
5 : **if** $(m^*, t^*) \notin R_i \wedge v == \mathsf{true}$
6 : **return** 1
7 : **else return** 0

$\mathsf{OGen}()$

1 : $k \leftarrow k + 1$
2 : $K^k_{\mathtt{m}} \leftarrow \mathsf{KeyGen}_{\mathtt{m}}(1^\lambda)$
3 : $R_i \leftarrow \emptyset$

$\mathsf{OMac}(i, m)$

1 : $t \leftarrow \mathsf{Mac}(K^i_{\mathtt{m}}, m)$
2 : $R_i \leftarrow (m, t)$
3 : **return** t

Figure 2.6 – The multi-instances SUFsecurity experiment.

Definition 2.15 (mi-SUF security). *A* MAC *scheme* $\mathsf{MAC} = (\mathsf{KeyGen}_{\mathtt{m}}, \mathsf{Mac}, \mathsf{Verif})$ *is said to be* mi-SUF *secure if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that:*

$$\mathsf{Adv}^{\text{mi-SUF}}_{\mathcal{A},\mathsf{MAC}}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\mathrm{SUF}}_{\mathcal{A},\mathsf{MAC}}(\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)\,.$$

We denote $\mathsf{Adv}^{\text{mi-SUF}}_{\mathsf{MAC}}(\lambda)$ *the corresponding advantage.*

The HMAC **function.** The HMAC function ([BCK96]) is a popular hash based message authentication code. Given a hash function $H : \{0,1\}^* \to \{0,1\}^c$, built on a compression function with bloc size b bytes, then:

$$\mathsf{HMAC}(k, m) = H(k^* \oplus \mathtt{opad} || H(k^* \oplus \mathtt{ipad} || m)),$$

where `ipad` is set to $0x36^b$ that is, the byte $0x36$ repeated b times, $\mathtt{opad} = 0x5c^b$ and k^* is defined as k padded with zeros to reach b bytes, if $\mathsf{length}(k) \leq c$, else k^* is $H(k)$ padded with zeros to reach

b bytes. This construction has been proven a PRF as long as the underlying compression function is a PRF ([Bel06]). The reduction from PRF security to EUF is given in [BKR00] (Theorem 2.7). Some more reductions, particularly from PRF to SUF are given in [AHM+14]). The security of HMAC can then be extended to the multi instance setting as explained in [PR18a].

2.3.4 Authenticated Encryption with associated data

An authenticated encryption with associated data allows to encrypt a message such that the ciphertext's integrity is guaranteed, and to attach to the message some additional data that are only protected in integrity. We refer to [Rog02] and [RBBK01] for the classical security definitions for AEAD and to [BN00] for the relations between the different notions of security.

Definition 2.16 (Authenticated encryption with associated data)**.** *An authenticated encryption with associated data* AEAD *associated with a key space $\mathcal{K}$, a message space $\mathcal{M}$, an associated data space $\mathcal{A}$ and a cipher text space $\mathcal{C}$, is defined by the following algorithms:*

- KeyGen : $1^\lambda \to \mathcal{K}$ *which on input a security parameter 1^λ, outputs a secret key* k*;*
- Enc : $\mathcal{K} \times \mathcal{M} \times \mathcal{A} \to \mathcal{C}$*, a probabilistic algorithm which, on input a secret key* k*, a message m and associated data a, outputs a ciphertext c (the associated data are usually not included in the cipher text and the way they are transmitted is not part of the model);*
- Dec : $\mathcal{K} \times \mathcal{C} \times \mathcal{A} \to \mathcal{M} \times \perp$ *which on input a secret key* k*, a ciphertext c and additional data a, outputs a message m or $\perp$ if the decryption was unsuccessful.*

An intuitive way to obtain AEAD is to compose an encryption scheme with a MAC scheme in an Encrypt-then-Mac fashion. The scheme ECIES (based on DHIES ([ABR01]), described in [Sho01] and standardized in [Sta06], or [EI04]) provides authenticated encryption with associated data.

2.3.5 Key Derivation Function

We follow [Kra10] for the definition of key derivation functions. First one needs to define a source of keying material. Informally, the source is a secret value (k) from which one would like to obtain a new key or even several keys. However, some side data (α) about this secret input may be available to an evesdropper for instance. This a formalized in the definition of a source of keying material.

Definition 2.17 (Source of keying material)**.** *A source of keying material* SKM *is a two value probability distribution (k, α) generated by an efficient probabilistic algorithm.*

Definition 2.18 (Key Derivation Function)**.** *A key derivation function* KDF *is a function that, on input a source of keying material SKM a length value ℓ, an optional salt value* XTS *and an optional context information ctx, returns a string of ℓ bits.*

As they are optional, the context and the salt can be set to O or to any constant value. We denote by $\mathcal{S}$ the salt space. What is expected from such a function is that, if given enough entropy *i.e.* a "good" source of keying material, then it should return a secure ℓ-bit key, that is, the output shall be indistinguishable from an ℓ-bit random string. This is formalized by the experiment given in Figure 2.7.

$Exp^{\mathsf{KDF}}_{\mathcal{A},\mathsf{KDF}}(\lambda, \mathsf{SKM})$

1 : $k, \alpha \leftarrow_{\$} \mathsf{SKM}$
2 : $\mathsf{XTS} \leftarrow_{\$} \mathcal{S}$
3 : $\ell^*, ctx^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KDF}}(k,\cdot,\mathsf{XTS},\cdot)}(\alpha, \mathsf{XTS})$
4 : $b \leftarrow_{\$} \{0,1\}$
5 : $k_0* \leftarrow \mathsf{KDF}(k, \ell^*, \mathsf{XTS}, ctx^*)$
6 : $k_1 * \leftarrow_{\$} \{0,1\}^{\ell^*}$
7 : $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KDF}}(k,\cdot,\mathsf{XTS},\cdot)}(\alpha, \mathsf{XTS}, k^*_b)$
8 : **return** $b = b'$

$\mathcal{O}_{\mathsf{KDF}}(k, \ell, \mathsf{XTS}, ctx)$

1 : **return** $\mathsf{KDF}(k, \ell, \mathsf{XTS}, ctx)$

Figure 2.7 – KDF security experiment.

Definition 2.19 (KDF security)**.** *A* KDF *function* KDF *is said to be secure with respect to a source of keying material* SKM *if, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that:*

$$\mathsf{Adv}^{\mathsf{KDF}}_{\mathcal{A},\mathsf{KDF}}(\lambda) = \Big|\Pr\left[\mathsf{Exp}^{\mathsf{KDF}}_{\mathcal{A},\mathsf{KDF}}(\lambda, \mathsf{SKM}) = 1 | b = 0\right] - \Pr\left[\mathsf{Exp}^{\mathsf{KDF}}_{\mathcal{A},\mathsf{KDF}}(\lambda, \mathsf{SKM}) = 1 | b = 1\right]\Big| \leq \mathsf{negl}(\lambda)\,.$$

We denote $\mathsf{Adv}^{\mathsf{KDF}}_{\mathsf{KDF}}(\lambda)$ *the corresponding advantage.*

The HKDF **function.** Hugo Krawczyk proposes in [Kra10] a construction for a key derivation function based on HMAC, called HKDF. It is composed of an extraction part, $\mathsf{HKDF}_{extract}$ that is used to extract a pseudo-random key from the input keying material k, if the latter is not already uniformly random or pseudo-random, and an expand function, HKDF_{expand} that derives several additional pseudo-random keys if needed. Let c be the output size of the hash function on which the HMAC construction is based, then:

$$\mathsf{HKDF}(k, \ell, XTS, ctx) = K(1)||K(2)\ldots||K(r), \text{ where } r = \lceil \ell / r \rceil,$$
$$PRK = HMAC(XTS, k),$$
$$K(1) = HMAC(PRK, ctx||0)),$$

and for $i \in [\![1, r]\!]$:

$$K(i) = HMAC(PRK, K(i-1)||ctx||i),$$

where $K(r)$ is truncated to its first $\ell \mod r$ bits.

2.4 Key exchange protocols

A key exchange is a protocol between - classically - two participants, traditionally denominated Alice and Bob. Alice and Bob exchange messages so as to share, at the end, a common secret data,

typically a symmetric key. This kind of protocol is particularly useful when a new symmetric key: a session key, needs to be shared each time Alice and Bob wish to start a conversation, *i.e.* want to start a new session. However, such a protocol is useless if Alice is not sure that the key she computes is only shared with Bob. Hence a secure key exchange protocol should ensure that no one, except the intended peer, learns anything about the newly shared key. This is the goal of authenticated key exchange. We detail in the following the definitions and the main security models of the key exchange literature.

2.4.1 How secure is your session key ?

As for any cryptographic primitive, an efficient security analysis needs to identify the security goals of the primitive. In the domain of key exchange, the most intuitive expectation is that no information about the session key leaks during the protocol execution. This has been formalised by Bellare and Rogaway in the seminal work [BR94], as an indistinguishability experiment. In this paper, the authors introduce the first security model for authentication and key exchange protocols, as we detail in subsection 2.4.2. But, with the increasing use of cryptographic key exchange, various more demanding requirements have emerged.

Known Keys Attacks. Resistance to known keys attacks, as defined in [BM99b], stipulates that the knowledge of one (or several) session key does not leak information about the other sessions keys. This is another basic requirement for a key exchange: the session keys should be computationally independent.

Key Compromise Impersonation (KCI) If Alice's long term key leaks to the adversary, nothing can prevent him to impersonate Alice. However, there is another form of impersonation that needs to be considered: does the knowledge of Alice 's secret allows $\mathcal{A}$ to impersonate any other uncorrupted party to Alice. This can be seen as a reverse form of impersonation and is called Key Compromise Impersonation (KCI, [KBB17], [BJM97]).

Unknown Key Share Attack (UKS). This attack is subtle as the adversary, we call him Evil, does not try to learn the session key. His goal is to make Alice think she communicates with him while her messages are in reality read by Bob. Bob identifies the messages he receives as coming from Alice - which is true - and, logically thinks they are addressed to him, when they were dedicated to Evil. If Alice is Bob and Evil's boss and the message is "you are fired" this can have dramatic consequences. A UKS attack breaks the assumption that if you receive a message from someone, then this message was intended to you.

Perfect Forward Secrecy (PFS). This notion cares about an adversary that registers the conversation and accesses the long term key after the key exchange is over. The definition in [MOV96] simply states: "A protocol is said to have perfect forward secrecy if compromise of long-term keys does not compromise past session keys." This says nothing about the power of the adversary. In [Kra05], Krawczyck defines weak PFS (wPFS) as the original PFS but only for sessions in which the adversary did not actively interfered. He shows that no two rounds implicitly authenticated protocol can achieve full PFS, but that it can verify wPFS. PFS has received a lot of interest from the late 90's, mostly for signature schemes (see [BM99a], [Kra00], [AR00], [IR01], [MMM02], [KR03], or [BSSW06]). Forward security in symmetric cryptography was studied in [BY03] while public

key encryption has been advised in [CHK03] for instance and FS in authenticated key exchange is considered in [DVW92] and was recently explored in [ACF20].

Post-Compromise Security (PCS). Often seen as a counterpart of PFS, Post-Compromise Security evaluates the possibility for a protocol to establish secure session keys after the long term secrets have been compromised. This notion has been formally defined and studied in [CCG16]. The authors differentiate a weak form of PCS, where the adversary is given access to a long term key interface (for instance a signature oracle) but not to the key itself. The strongest PCS admits a compromise of the long term secret. The adversary can interfere in all but the last exchange before a secure session key shall be established. A main result of this work is that no stateless key exchange protocol can achieve full PCS.

Future secrecy, healing and backward secrecy also deal with the security of future keys. The term self-healing appeared in [CJZ11] to designate the property given by a continuous refresh of the secret material used to compute subsequent session keys. If an attacker accesses the secret material, he can only eavesdrop on the corresponding session key. If he misses this opportunity, security is regained. According to one of the designer of Signal[3], future secrecy means that "a leak of keys to a passive eavesdropper will be healed", which is equivalent to the definition of self healing[4]. In [BSJ+17], the authors refer to backward secrecy as "the knowledge of sender's secrets - state and keys - at the current time period can not be used to distinguish keys generated (at some near point) in the future from random string".

A "major detail" in the above definitions is the amount of passivity required from the adversary (one message, one exchange or more). As shown in [CCG16], if the adversary is fully passive, then the future secrecy is already covered by PFS. If the adversary can not use his knowledge of the long-term secrets to introduce known randomness, or to fool the authentication for instance, then there is no difference in corrupting the long-term secret before or after the session. If the adversary is not required to be fully passive, PCS differs from PFS in that it concerns the authentication as well as the confidentiality of the future key.

As we will see in chapter 3, in the context of Ratcheted Key Exchange (RKE), each ratchet can be considered as a sub session and then PFS and PCS is to be understood relatively to a state corruption besides the long term key corruption.

2.4.2 A formal model

In the seminal work [BR94], Mihir Bellare and Philip Rogaway formalized the notions of authentication and key exchange. They introduced a formal model and useful definitions. This has been the beginning of a fruitful literature on AKE. Most of the formalism we recall here is inherited from their work, however augmented with further evolutions.

The environment. The model considers a set of participants I, who can run the protocol. Each participant i is given a long term key. It can be a (public, private) key pair (pk_i, sk_i), or a symmetric key sk_i (the original model is in the symmetric key paradigm. It has been extended to the public key setting in [BJM97]). Two participants perform the protocol as intended until they reach a status `accept` or `reject` (before they reach one of them, their status is undetermined (written as $*$)).

[3] Trevor Perrin, as quoted in [CCG16], the original reference on GitHub is not available any more.

[4] In their description of the Double Ratchet properties [MP13], the authors clearly mention that their future secrecy is equal to the self healing.

The primitive: a formal protocol. A protocol Π can be seen as an efficiently computable function that takes as input: a security parameter λ, the identity of the owner of the session $i \in I$, the intended partner $j \in I$, the secret of the owner $sk_i \in \mathcal{K}$, an history of the messages exchanged so far, also called the transcript $t \in \{0,1\}^*$, the random tape of the owner $r \in \{0,1\}^\infty$. Then $\Pi(i, j, sk_i, t, r)$ returns the next message m to be sent when following the protocol description, together with the status $\mathtt{st} \in \{\mathtt{accept}, \mathtt{reject}, *\}$ of the owner, and a potential private output p (commonly the session secret key). This formalism is identified as message driven protocols in [CK01b]. The ℓ^{th} session involving the participant i (the owner of the session) and the participant j (the intended peer) is identified, on the administrative side of the model, as a triple $s = (i, j, \ell) \in I \times I \times \mathbb{N}$. A session s is associated to a tuple (i, j, $role$, sid), where $role \in \{initiator, responder\}$ is the role dedicated to i and sid is a session identifier. Session identifiers are defined to bind the messages to their corresponding session. They can correspond to a practical value set by the protocol or be explicitly defined by the model as we will see later. Each session s is associated to a session oracle π_i^s. The creation of these oracles can be formalized by a Create oracle (also often called Init oracle) that takes as input a triple (i, j, $role$), initialises an oracle π_i^s and returns the corresponding session identifier sid (that may be $\perp$). This oracle keeps in memory the local state for the session, namely the owner identity, $\pi_i^s.id = i$, the intended partner's identity $\pi_i^s.peer = j$, the session key $\pi_i^s.sessionkey = k_i^s$ (initialized to $\perp$), the transcript $\pi_i^s.trans = t_i^s$ (initialized to $\perp$), and the random tape $\pi_i^s.rand = r_i^s$. The execution of the protocol is controlled by an oracle Send. On input a tuple (i, j, s, m), where m is an incoming message for the session played by π_i^s, it returns (m', $\mathtt{st}$) such that (m', $\mathtt{st}$, p) = $\Pi(i, j, sk_i, t_i^s||m, r_i^s)$. The state of the corresponding π_i^s is updated if necessary (for instance the transcript is augmented, the session key field receives the private output p, ...). To start an initiator session, the incoming message is set to $init$.

The adversary. The adversary $\mathcal{A}$ can create sessions between the participants he pleases, by calling the Create oracle. He also controls the message delivery with the oracle Send. In that sense, he can reorder or drop messages. As we will see later, this is not always possible in more complex multi stages models for instance (*cf.* section 2.4.3). The adversary can be honest but curious, which means he relays the messages as expected and only tries to grab information from what he sees. He can also try to introduce his own information or modify the communication. Or he can be given more power. In [BR94], $\mathcal{A}$ has access to a RevealSessionKey oracle that, when queried with a triple (i, j, s), returns the corresponding session key if the session π_i^s's status is $\mathtt{accept}$ and $\perp$ otherwise.

In [CK01b], Ran Canetti and Hugo Krawczyck extend the Bellare-Rogaway model of [BR94]. Their model is simply identified as the CK model. Their work increases the power of the adversary by giving him access to new oracle queries[5]. Firstly, in addition to the RevealSessionKey oracle, $\mathcal{A}$ can call a RevealState oracle that, when queried on a session (i, j, s), returns the local state of participant i for this particular session. The local state may contain for instance the random coins (the ephemeral Diffie Hellman elements for instance) needed for the computation of the session key. Secondly, with a Corrupt(i) query, $\mathcal{A}$ gets the complete state of a participant i. He obtains i's random coins, the local state for every session i is engaged in, and most of all, he obtains his long-term key. This definition - even if realistic (one can easily imagine that an adversary that can access to the memory where the long term secrets are registered also has access to the memory where the local states are registered, as the former should be more secure than the latter) - is tricky

[5] These new queries were actually already defined in [BCK98]. However the model proposed in this paper is based on the simulation paradigm that brings composability but results in less intuitive proofs, hence we focus on game based proofs. The Corrupt query also appeared in [BPR00]. We decide to focus on [CK01b] that appeared as a bigger step in the evolutions of AKE models.

because it demands to consider whether a session is expired or not. The local state of an expired session is erased and so, never revealed with a Corrupt query.

In their extended version of the CK model [LLM07] (now simply identified as the eCK model), La Macchia *et al.* consider a RevealLongTerm query only reveals the long term secrets. This is not restrictive as the Corrupt information, in the sense of the CK model, is still accessible by combining a RevealLongTerm and RevealState queries. We will adopt this version in our own contribution. Another difference lies in the definition of the RevealState. In the CK model, what is included in the state is not formally defined. It is protocol dependent. Does it contains all random coins (for instance those used for signature or encryption if needed)? Or only some specific data? An analysis can be conducted with the hypothesis that the state is empty. Hence the single CK model can cover a large range of practical security. La Macchia *et al.* replaces the RevealState by a more precise RevealEphemeralKeys. The formalisation of these corruption queries will be of prime importance when turning to the ratcheted key exchange. To keep trace of this queries, the state of a session oracle will be augmented with associated "model-specific" flags, that we identified with a `typewritter` font, for instance π_i^s.`revSessionKey`, π_i^s.`revState`. As the corruption of the long term key is not session specific, we consider it is recorded in a global state for the participant, that can be written as π_i. The associated flag is then π_i.`corrupt`. Whatever the model, the adversary is not given unlimited access to these oracles, otherwise he could win trivially. Restrictions are formalised in a freshness predicate, which we detail in section 2.4.2.

Matching sessions. A crucial tool for the authentication are the matching sessions. This notion measures whether the participants really talked to each other. The definition of matching session may vary from a model to another. In the original definition of [BR94], matching sessions are defined on matching conversations. Consider that the transcript of a session records the succession of messages sent and received, timely ordered. Then a session s' is matching a session s if its transcript $\pi_j^{s'}.trans$ corresponds to the transcript $\pi_i^s.trans$. The correspondence is not a mere equality for two reasons: firstly for an initiator session s the transcript will be prefixed with an $init$ label that will not appear in the responder transcript. Secondly, suppose an odd number of communications, and consider an initiator session s. Then a session s' is matching s even if its transcript does not contain the last message sent by the initiator session π_i^s. All one needs to be sure is that the messages received by the initiator π_i^s are messages created by $\pi_j^{s'}$ after receiving a message from π_i^s.

The definition of matching session based on the transcript is commonly adopted in the key exchange literature. However in [CK01b] or even [BPR00], the matching is based on session identifiers: two sessions $(i, j, role, sid)$ and $(j, i, \overline{role}^{6}\ sid')$ are matching if $sid = sid'$. While the definition seems more adapted to the case where the protocol explicitly defines a session identifier, the author of [BPR00] point out that, if it is not the case, it may be set to the protocol flow for instance. The same way, in [Kra05], when analysing the protocol HMQV in the CK model, the author sets the session identifier to be the couple $(sent, rcv)$ of messages sent and received by the session and refines the definition by saying that "the session $(j, i, rcv, sent)$ (if it exists) is said to be matching the session $(i, j, sent, rcv)$"[7]. As the definition based on the transcript appears as quiet natural, we identify sid with $(rcv, sent)$ and formalise it as follows:

[6] An overlined value classically denotes the opposite (the complement) value. Here if $role = initiator$, then $\overline{role} = responder$ and reversely.

[7] A third way of defining matching is through a matching function, as in [BR95]. However it is dedicated to the three party case and we will not develop it here.

Definition 2.20 (matching session). *A session $(id_{own}, id_{peer}, (send, rcv), role)$ is said to be matching $(id'_{own}, id'_{peer}, (send', rcv'), role')$ if both sessions are completed and $id_{own} = id'_{peer}$, $id_{peer} = id'_{own}$, $send = rcv'$, $rcv = send'$, and $role = \overline{role'}$.*

The separation between messages sent and received in the transcript enables to refine the definition. In [CF12] the notion of origin session is introduced, that will be a key element for the definition of a PFS secure key exchange.

Definition 2.21 (origin session). *A session $s' = (id_{own}, id_{peer}, (send, rcv), role)$ (possibly incomplete) is said to be an origin session for $s = (id'_{own}, id'_{peer}, (send', rcv'), role')$ if $send' = rcv$.*

The idea is that, to achieve full PFS, we only need to ensure that the messages received by the test session where not controlled by the adversary, hence, that they were created by a legitimate oracle.

Authentication. Mutual authentication is studied as a separate goal in [BR94]. In the authentication process, the goal of $\mathcal{A}$ is to make an oracle $\pi^s_{i,j}$ reach an `accept` status while having no matching sessions. Their definition of authenticated key exchange is too strong and restrictive as it requires an AKE to be a secure key exchange on top of an authentication protocol. On the contrary, the author of [BR95] claim that authentication is not a goal of security for key distribution, and, traducing this view, one can say that the important point is not to be sure that the intended peer was on-line at the time the key exchange took place, but only that the key is shared with this intended peer and only with him. This is even more verified in the case of asynchronous key exchanges, where both participants do not need to be on line at the same time. Finally, in most of authenticated key exchanges, the authentication part is implicit: it is implied by the fact that the two participants derive the same key and hence is only verified when the key is used.

Key exchange security. Considering the key exchange part, the expected security is that the adversary does not recover the secret key, but, more than that, learns no bit of information about it. Once again, this can be formulated as an indistinguishably experiment $\mathsf{Exp}^{\mathrm{AKE}}_{\mathcal{A},\Pi}(\lambda)$. The adversary runs the protocol Π. The challenger creates and plays with the oracles π^s_i. At some point he queries the challenger for a Challenge on a specific session (i^*, j^*, s^*) (that should corresponds to some oracle execution $\pi^{s^*}_{i^*}$). This session is identified as the challenged or test session. He is then given, depending on a random bit b sampled by the challenger, either the corresponding correct session key, or a random key. This event is recorded with a flag $\pi^{s^*}_{i^*}$.`test`. The adversary may continue to query other oracles afterwards, however he is only permitted one challenge query. He must *in fine* decide which version he received and release a bit b'.

Definition 2.22 (Authenticated Key Exchange (AKE)). *A protocol Π is said to be a secure authenticated key exchange (relatively to the above security model) if, for all PPT adversary $\mathcal{A}$ making at most q queries and opening at most q_s sessions:*

- *any two sessions π^s_i and π^t_j that are matching and that accept compute the same session key;*
- *there exists a negligible function* negl *such that:*

$$\mathsf{Adv}^{\mathrm{AKE}}_{\mathcal{A},\Pi}(\lambda) \leq \mathsf{negl}(\lambda)\,.$$

We denote $\mathsf{Adv}^{\mathrm{AKE}}_{\Pi}(\lambda, q, q_s)$ the corresponding global advantage.

The freshness of a session. In the following, we consider a test session s =(i, j, $role$, $sent$, rcv) and π_i^s the associated oracle. If he plays the game described above, the adversary has an easy solution to win: he first reveals the session key of a session oracle π_i^s, then challenges this session. To prevent those attacks that are unavoidable, a model sets natural restrictions, often called the freshness conditions. A session key can be tested only if the session is fresh. In the model of Bellare and Rogaway, a session π_i^s is fresh if it has accepted, it has not been queried with a RevealSessionKey, and there is no session π_j^t, matching π_i^s, that has been queried with a RevealSessionKey. The freshness crafts the security covered by the model and the more power the adversary is given, the trickier is the definition of the freshness. We give some examples of the relation between the oracles given to $\mathcal{A}$, the freshness restrictions and the security properties.

Know Key Attacks. Resistance to Known Key Attack is formalised by giving the adversary the access to the RevealSessionKey oracle for all sessions but the test session and its matching peer, if it exists.

Perfect Forward Secrecy. To capture PFS, one needs to give the adversary the possibility to compromise the long-term key of the test session, and of its matching peer (if it exists), once the session is completed. A basic restriction is that the adversary is not allowed to compromise both the long-term and the state of the owner of the session or of its matching peer. However this last part is only valid if there exists a peer session oracle that indeed send the message to π_i^s *i.e.* , using the definition of origin session recalled in section 2.4.2, if there exists an origin session. More than that, depending on the conditions under which $\mathcal{A}$ is authorized to corrupt a participant, one obtains different flavours of PFS. With no additional requirement, Krawczyk showed in [Kra05] that no two rounds protocol (with implicit authentication) can achieve PFS. He introduces the weak-PFS definition where $\mathcal{A}$ has access to the RevealLongTerm oracle only if he did not interfered in the protocol during its execution. This can be written as: $\mathcal{A}$ is given access to the oracle RevealLongTerm(j) only if there exists an origin session for s. This is the eCK^w model.

In the full PFS case (eCK-PFS), this condition is changed to the following: if there exists no origin session (if the message received by i were controlled by $\mathcal{A}$) then s is fresh if $\mathcal{A}$ did not query a RevealLongTerm(j) before the session was complete. Note that no restriction is needed on the owner's side. For the session s to accept, the oracle must have performed the entire protocol. The adversary can modify the messages sent from i to j, however, he will not control the ephemeral generated by π_i^s without a RevealState query.

Key Compromise Impersonation. Turning to KCI, the adversary shall not be prevented to compromise the owner i's long-term key before the execution of the session. This is the case in the eCK, eCK^w and eCK-PFS models.

Post Compromise Security. We deeply examine this aspect in the next Chapter, section 3.3.

As we see, freshness must be carefully balanced to exclude unavoidable attacks while giving enough freedom to the adversary to capture the desired properties.

2.4.3 Beyond AKE

The AKE model as described above covers the basic but fundamental goal of sharing a secure key between two participants. It does not, however, consider the way this key is (then) used, even though it can dramatically harm the security. We mention two different ways of taking this issue into account. The composition, that states the conditions for a session key computed by a key exchange to be used to provide an authenticated and secrete channel for instance. And the ACCE model, that proposes a complete security model that includes the key exchange in a broader protocol. On another side, multi stage key exchange, that computes several keys for several functions, do not

fit neither into the AKE model. We introduced the definitions specific to the multi-stage context, as a first step toward ratcheted key exchanges that we develop in the next chapter.

Secure Channels and the ACCE model. We first recall an attack described in [CK01a] (Appendix A, this attack is attributed to Rackoff): imagine a key exchange protocol in which the first participant that computes the session key k, exposes k as soon it receives a well computed tag $\mathsf{MAC}(k, 0)$. If studied in the original model designed by Bellare and Rogaway, the key k can be shown secure, because the adversary is compelled to provide its guess bit just after receiving its challenge value. He has no chance to foul the challenger by sending a MAC value. However, this protocol can not be securely combined with a MAC scheme. Further models hence let the adversary play with the challenger after the Challenge query. But this example shows that a secure use of the session key is not trivial. In [CK01b], Canetti and Krawczyk introduce the notion of secure channel. Informally, a secure channel is a link between two participants where data can be transmitted in a secrete and authenticated manner. The authors show that a session key computed by a key exchange protocol shown secure in the CK model, can be combined with secure MAC and a secure encryption scheme to provide such a channel. They provide a security reduction where an adversary that could foul the authentication (respectively the secrecy) of the channel, would break the security of the key exchange. The composability of key exchange protocol, in a game-based formalism[8] is extended in [BFWW11], to any symmetric key protocol. The composition goes as follow: consider a key exchange protocol Π_{ke} that provides session keys relatively to a key distribution $\mathcal{K}$. Consider a symmetric protocol Π_{sym}, defined on the key distribution $\mathcal{K}$. The composed protocol $\Pi_{\mathsf{ke,sym}}$ consists in first running Π_{ke} then, if a session key is accepted, running Π_{sym} with the session key. The set-up (more particularly the key generation) for the composed protocol is the set up of the key exchange protocol.

On the security side, let $\mathsf{Exp}^{\mathrm{SEC}}$ be a security experiment for the symmetric primitive. Then one can define a composed security experiment $\mathsf{Exp}^{\mathrm{AKE,SEC}}$ in which an adversary $\mathcal{A}$ can interact simultaneously with several executions of the key exchange and the symmetric protocol, with the symmetric keys being those derived from the AKE. The final goal of $\mathcal{A}$ is to break the security of the symmetric protocol, not the key exchange. The authors prove a general reduction result:

Theorem 2.3. *Let Π_{ke} be a secure authenticated key exchange[9], relatively to $\mathcal{K}$, and Π_{sym} be a* SEC *secure protocol. Then for any PPT adversary $\mathcal{A}$, there exists PPT adversary $\mathcal{B}, \mathcal{C}$ such that:*

$$\mathsf{Adv}^{\mathrm{AKE,SEC}}_{\mathcal{A},\Pi_{\mathsf{ke,sym}}}(\lambda) \leq n^2 q_s \cdot \mathsf{Adv}^{\mathrm{AKE}}_{\mathcal{B},\Pi_{\mathsf{ke}}}(\lambda) + \mathsf{Adv}^{\mathrm{SEC}}_{\mathcal{C},\Pi_{\mathsf{sym}}}(\lambda),$$

where n is the maximum number of participants and q_s the maximum number of sessions per ordered pair of participants.

The reduction coefficient comes from the fact that one does not know which symmetric key will be targeted by the adversary. As a consequence, one has to replace, step by step, all the session keys by random elements - this is called a hybrid argument -, which amounts to playing $n^2 q_s$ times the AKE game, before calling the symmetric security independently from the key exchange.

[8] Composability of key exchange has also naturally been studied in the simulation paradigm, for instance in [CK02] or [Sho99], as it is one of the great advantage of this framework. However, as noted earlier, simulation based proofs are less intuitive and often come with stronger restrictions that prevent efficient secure realisations.

[9] The original paper considers security relatively to the original Bellare-Rogaway model, while claiming it can be extended to eCK without difficulty.

This modular approach, first the key exchange, then the communication channel is often adopted, for its modularity and simplicity, and is the one adopted for the analysis of Signal, on which we rely in our first contribution. However, key exchange protocols that use some exchanged values within the protocol can not be proven secure with this approach. The Transport Layer Security protocol (TLS) is widely used to establish Internet channel. It mainly consists in two phases: a handshake phase (the key exchange part) and a record layer (communication phase). In earlier versions (for instance TLS-DHE 1.2, as TLS 1.3 is now standardised), a main obstacle to prove the security was that a last message, the server finish, was encrypted using the freshly computed session key, preventing any proof in the traditional key indistinguishability model. Therefore, while studying the security of this protocol, the authors of [JKSS12] define the notion of authenticated and confidential channel establishment (ACCE). The idea behind ACCE, is to keep a modular approach between an authenticated part and a secrecy part, but with a weaker requirement on the key exchange. In the ACCE model, one does not require the indistinguishability of the session key, but only study the indistinguishability of the ciphertext produced with this key.

Multi-stage. TLS is not only a textbook case for the ACCE security. It also provides session resumption. This mechanism enables two participants to derive a new session key without going through the whole initial key agreement. Instead, they rely on a common key previously computed within a full handshake step. The complete key exchange is now composed of two stages, first a handshake, then a session resumption, both resulting in a different session key. This notion of multi-stage key exchange has been formalised by Fischlin and Günther in [FG14] to provide an analysis of Google's Quic protocol. We will detail some aspects of their model as an introduction to the ratcheted key exchange models studied in section 3.3).

A stage gathers several steps of the protocol that result in a session key. This stage specific session key can be further used in the protocol (to derive the next key for instance) or with an external cryptographic primitive. Then studying such a protocol, one should ensure that each stage specific session key is secure. Contrary to the single stage classical models, the adversary can choose among a multiplicity of targets within each session. In the model administration, a stage is an additional information that is recorded within the local state of the session. In our formalism, this translates in a field $stage \in \mathbb{N}$ that is recorded by each oracle π_i^s. The number of stages per session is generally bounded by a value n_{st}. Now, as several keys are computed, the single session key field sk in a session oracle state is replaced by a set SK gathering all $k_t = SK[t]$ established in stage t. The flags that record the reveal or challenge of a session key have to be adapted accordingly.

Beyond those administrative changes, the proposition of Fischlin and Günther tackles the notion of the scheduling of the different stages that will also be predominant in the ratcheting process. Firstly, as a stage-t key may be used in a further stage $s > t$, the model introduces a stop and go mechanism: if a Send query leads to the acceptance of a stage key, then the protocol halts, for the adversary to query a Challenge if he wants to. If he does, the value released in the test is set as the stage key value for the rest of the execution. Secondly, if a key at some stage $t + 1$ depends on the session key of the previous stage $SK[t]$ (key dependence), then the adversary cannot reveal $SK[t]$ before $SK[t + 1]$ is established. This key dependence weakens the security model. However it still prevents simple derivation (for instance $SK[t + 1] = \mathsf{h}(SK[t])$ for some hash function h). Finally, the forward secrecy, if it is provided by the protocol, may not be accessible for the early stage. A protocol is t-forward secure if it guaranties PFS only from stage t (included).

An interesting result is about composability: adopting the formalism of [BFWW11], they show that key independence enables the composability with any symmetric scheme. To conclude on TLS,

an analysis of the 1.3 version of the handshake protocol has been conducted in this multi-stage model in [DFGS20].

2.5 PRF, hash functions and random oracle

In this section, we explore the gap - and the relations - between formal security definitions related to pseudorandom and hash functions, and an idealisation, the random oracle model, that is often adopted to design efficient protocols.

We denote $\mathsf{Func}(\mathcal{D}, \mathcal{R})$ the set of all functions with domain $\mathcal{D}$ and range $\mathcal{R}$, and $\mathrm{FF}(\mathcal{K}, \mathcal{D}, \mathcal{R})$ the set of all function family with parameter (key) in $\mathcal{K}$, domain $\mathcal{D}$ and range $\mathcal{R}$. In the following, we will note $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ a function family in $\mathrm{FF}(\mathcal{K}, \mathcal{D}, \mathcal{R})$ (and call it a function, by ease of language).

Definition 2.23 (Pseudorandom function)**.** *Let $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be an efficiently computable function family. It is said to be pseudorandom if, for all probabilistic polynomial time $\mathcal{A}$:*

$$|\Pr\left[k \leftarrow_{\$} \mathcal{K}, \mathcal{A}^{\mathcal{O}_{\mathsf{f}(k,\cdot)}} \to 1\right] - \Pr\left[g \leftarrow_{\$} \mathsf{Func}(\mathcal{D}, \mathcal{R}), \mathcal{A}^{\mathcal{O}_g} \to 1\right]| \text{ is negligible.}$$

If the adversary is not given a complete oracle access to f_k or g but only receives either the set $\{r_1, \ldots, r_q, f(k, r_1), \ldots, f(k, r_q)\}$ (r_i's uniform random values in $\mathcal{D}$) or $\{r_1, \ldots, r_q, u_1, \ldots u_q\}$ (the u_i's uniform random values in $\mathcal{R}$), then f is said to be a weak-pseudorandom.

Hash functions. Let length function $\ell_{\mathsf{out}} : \mathbb{N} \to \mathbb{N}$, be a length function. Basically, a hash function is a compressing function $h : \{0,1\}^* \to \{0,1\}^{\ell_{\mathsf{out}}}$. Theoretically, one often considers families of keyed hash functions $\mathsf{h} : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$. A keyed hash function h with domain $\mathcal{D}$ and range $\mathcal{R}$ is determined by a generation algorithm Gen_h that, on input a security parameter λ, outputs a valid key $k \in \mathcal{K}$, that provides a valid description for $h_k : \mathcal{D} \to \mathcal{R}$. As a consequence, a keyed hash function can be identified as a couple $H = (\mathsf{Gen}, h)$. A hash function is expected to be collision resistant as defined below.

Definition 2.24 (Collision resistant hash function.)**.** *A hash function $H = (\mathsf{Gen}, h)$ is said to be collision resistant if, for all probabilistic polynomial time $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:*

$$\Pr\left[k \leftarrow \mathsf{Gen}(1^\lambda), (x, x') \leftarrow \mathcal{A}(h_k, \mathcal{D}, \mathcal{R}) : x, x' \in \mathcal{D}, x \neq x' \text{ and } h_k(x) = h_k(x')\right] \leq \mathsf{negl}(\lambda)\,.$$

2.5.1 The Random Oracle Model (ROM)

The Random Oracle Model is an ideal environment where all the parties (including the adversary and the challenger) are given access to an ideal function, the Random Oracle (RO), which, on input a query, returns a perfectly random sequence of adequate length ℓ. It also keeps a record of past queries such that it returns the same result when queried twice (or more) on a same input x. Typically, the RO can be thought of as a huge table, gathering (query, output) couples, with no collision within the random sequences (up to the unavoidable birthday bound, $q^2/2^\ell$ where q is the number of queries). More formally, given a security parameter $\lambda \in \mathbb{N}$, $\mathcal{O}$ is a function sampled among all the possible functions mapping $\{0,1\}^*$ to $\{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$. The ROM methodology was introduced by

Bellare and Rogaway in [BR93], inspired by the work of Fiat and Shamir on signature schemes ([FS87], *cf.* subsection 2.6.2) and Goldreich, Goldwasser and Micali on random functions ([GGM84]). As practical as it may appear, the ROM has a main drawback: as no such random oracle exists in real life, a protocol proven secure in the ROM will be implemented with a real function — generally a hash function h — playing the part of the oracle. If a pseudorandom function f can be sampled and kept secret by a trusted third party who, when queried on x, answers $f(x)$, then the implementation of the oracle is possible (see [MNPV99] that studies a concrete solution). Most of the time however, the users need the function to be fully specified (directly available, not through a third party).

In [CGH98], Canetti, Goldreich and Halevi show the limits of the ROM. They prove that the ROM cannot be implemented by a single function, that would be valid for all protocols and, going further, they exhibit a signature and an encryption schemes that can be proven secure in the ROM but that cannot be secure when implemented with any real life function ensemble. This could be have been the death sentence of the ROM, but it has not, mainly because these examples are theoretical ("unnatural" as qualified by the authors). On the positive side, no practical protocol proven secure in the ROM has been shown insecure because of the implementation of the ROM by a real life hash function. The final discussion of [CGH98] is still relevant: as Canetti, it is reasonable to think that relying on an unproven ideal construction such as the Random oracle contradicts the principles of cryptography and proven security. But the fact is that the ROM provides efficient and, until now, secure practical construction that would not be possible otherwise. It seems not completely foolish to stand in a midway, saying that, while concentrating on designing protocols proven secure in the plain model, one can rely on the ROM as long as no other practical solution appears.

2.5.2 Correlation Intractability

Together with their criticism of the ROM, Canetti, Goldreich and Halevi propose [CGH98] a formalisation of the security one can expect from the random oracle, called correlation intractability. As for a random oracle, the sets of functions are determined by their output length:

Definition 2.25 (ℓ_{out} ensemble)**.** *Let $\ell_{\mathsf{out}} : \mathbb{N} \to \mathbb{N}$ be a length function. An ℓ_{out} ensemble is a sequence $\mathcal{F} = \{f_\lambda\}_{\lambda\in\mathbb{N}}$ of function families, such that, for any $\lambda \in \mathbb{N}$, $f_\lambda = \{f(k,\cdot) : \{0,1\}^* \to \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}\}_{k\in\{0,1\}^\lambda}$ is an efficiently computable function family.*

Correlation intractability is defined relatively to *evasive* relations. Informally, a binary relation $\mathcal{R}$ is said to be evasive if, given access to a random oracle $\mathcal{O}$, it is nearly impossible to find a witness x such that $(x, \mathcal{O}(x))$ satisfies the relation $\mathcal{R}$ (*i.e.* belongs to the language $\mathcal{L}_{\mathrm{R}}$ defined by R).

Definition 2.26 (Evasive binary relation)**.** *A binary relation R is said to be evasive relatively to a length function ℓ_{out} if, for every probabilistic polynomial time $\mathcal{A}$, there exists a negligible function* negl *such that:*

$$\Pr_{\mathcal{O}}\left[x \leftarrow \mathcal{A}^{\mathcal{O}} \wedge (x, \mathcal{O}(x)) \in \mathcal{L}_{\mathrm{R}}\right] \leq \mathsf{negl}(\lambda)\,.$$

Definition 2.27 (Correlation intractability)**.** *Let $\ell_{\mathsf{out}} : \mathbb{N} \to \mathbb{N}$ be a length function and $\mathcal{F}$ be an ℓ_{out} ensemble. Let R be a binary relation. $\mathcal{F}$ is correlation intractable relatively to R if, for every probabilistic polynomial time oracle $\mathcal{A}$ there exists a negligible function* negl *such that:*

$$\Pr_{k\in\mathcal{K}}[x \leftarrow \mathcal{A}(k) \wedge (x, f(k,x)) \in \mathcal{L}_{\mathrm{R}}] \leq \mathsf{negl}(\lambda)\,.$$

$\mathcal{F}$ is correlation intractable if it is correlation intractable relatively to any evasive binary relation.

A main result of [CGH98] is that there exists no correlation intractable ensemble. They extend their result to some specific case where the input length is bounded. This result can be interpreted as the fact that there exists no function family that can realise a generic random oracle. However, the interesting point is that some other results give some construction of correlation intractable function relatively to more restricted classes of binary relations (example [CCR15]). In [Oka93] the author assumes that there exists a function ensemble that is correlation intractable relatively to specific discrete logarithm computations relations, to prove the existential unforgeability of their signature scheme.

Definition 2.28 (Sparse binary relation)**.** *A binary relation* R *is said to be sparse with respect to length functions* $\ell_{\mathsf{in}}, \ell_{\mathsf{out}}$ *if there exists a negligible function* negl *such that, for all* $x \in \{0,1\}^{\ell_{\mathsf{in}}}(\lambda)$*:*

$$\Pr_{y \in \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}}[(x,y) \in \mathrm{R}] \leq \mathsf{negl}(\lambda)\,.$$

The difference between sparse and evasive relations is that the evasive definition measures the difficulty to find an input/output pair that verifies the relation relatively to a non-uniform adversary and this adversary can invoke several time the relation. In [CCR16], the authors give a construction for a function family that is correlation intractable relatively to sparse relation recognizable by a circuit of size bounded by a given polynomial p.

2.5.3 Correlated input security

Correlated input security is introduced by Goyal, O'Neill and Rao in [GOR11], in the context of hash functions. Three different degrees of security are considered: one wayness, unpredictability and pseudorandomness. The latter notion can be seen as a generalisation of the correlated robustness introduced by Ishai *et al.* in [IKNP03]. We focus on one-wayness as it is the version we will need later in our contribution.

Let $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be a function family. (Note that we keep the notation of general function family for consistency with the definitions we will give in chapter 5. However, to glue with the original definition, f can be thought of as a keyed hash function where the key is a public parameter, and is generated by a specific algorithm instead of being sampled from a key space.) Let $\mathcal{C}$ be a class of polynomial size circuits $\mathcal{D} \to \mathcal{D}$, such that each circuit shall have a sufficient min-entropy output distribution for uniform input distribution. Let $(C_1, \ldots, C_n) \subset \mathcal{C}$ be a tuple that, for any random $r \leftarrow\!\!\$\, \mathcal{D}$, $(C_1(r), \ldots, C_n(r))$ defines a tuple of correlated inputs.

The adaptive $\mathsf{Exp}^{\mathcal{C}\text{-aCI-ow}}_{\mathcal{A},f,\mathcal{C}}$ experiment:

- **Set-up.** The Challenger samples $k \leftarrow\!\!\$\, \mathcal{K}$, $r \leftarrow\!\!\$\, \mathcal{D}$ and sends k to $\mathcal{A}$.
- **Queries.** The adversary sends polynomially many queries $C_1, \ldots C_q$, $C_i \in \mathcal{C}$ for all $i \in [1, q]$, $q = poly(1^\lambda)$. The challenger answers with q values $f_i = f(k, C_i(r))$.
- **Invert.** The adversary outputs (x, j). The experiment returns 1 if $f(k, x) = f_j$.

In the selective version, the adversary is asked to output his queries before seeing the public parameter k (*i.e.* before accessing the full description of the hash function).

Definition 2.29. *A family of function* f *is said to be adaptive correlated-input one way with respect to a family of correlated input-circuits* $\mathcal{C}$ *if, for all probabilistic polynomial time* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\mathsf{Adv}^{\mathcal{C}\text{-aCI-ow}}_{\mathcal{A},f,\mathcal{C}}(\lambda, q) = \Pr\left[\mathsf{Exp}^{\mathcal{C}\text{-aCI-ow}}_{\mathcal{A},f,\mathcal{C}} = 1\right] \leq \mathsf{negl}(\lambda)\,.$$

2.5.4 Related Key Attacks

We now turn to property that can be considered as a "dual version" of the previous correlated input one wayness. Related key attacks appeared in the context of cryptanalysis of real block ciphers (see [BK09] for instance). It was latter formalised by Bellare and Kohno in [BK03]. Their work is mostly focused on PRP but the full version extends to PRF. The definition takes as a parameter a set Φ of key related derivating (RKD) functions $\phi : \mathcal{K} \to \mathcal{K}$, that will define the relations taken into account.

Definition 2.30 (Φ-RKA security)**.** *Let Φ be a set of RKD. A PRF $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ is said to be secure against Φ-restricted related key attacks (Φ-RKA secure) if, for all probabilistic polynomial time $\mathcal{A}$, there exists a negligible function* negl *such that:*

$$|\Pr\left[k \leftarrow_{\$} \mathcal{K}, \mathcal{A}^{\mathcal{O}_{f(RK(\cdot,k),\cdot)}} \to 1\right] - \Pr\left[g \leftarrow_{\$} \mathbb{F}(\mathcal{K},\mathcal{D},\mathcal{R}), \mathcal{A}^{\mathcal{O}_{g(RK(\cdot,k),\cdot)}} \to 1\right]| \leq \mathsf{negl}(\lambda)\ ^{10},$$

where $\mathcal{O}_{f(RK(\cdot,k),\cdot)}$ (respectively $\mathcal{O}_{g(RK(\cdot,k),\cdot)}$) takes as input a couple $(\phi, r) \in \Phi \times \mathcal{D}$ and returns $f(\phi(k), r)$ (resp. $g(\phi(k), r)$).

Bellare and Khono show that some class of relations are impossible, for instance, no PRF (or PRP) can be proven secure against RKA restricted to a class that contains a constant function. On the positive side, Bellare and Cash propose in [BC10] a PRF construction where the keyspace is a group, and that is secure against RKA where the key is operated on by any adversary specified group element, under the DDH assumption. Finally, in [BCM11], Bellare *et al.* model traditional block ciphers as RKA secure PRFs and look how to use this primitive to build other RKA secure primitives, such as signatures or encryption schemes.

In [GOR11], the authors prove the transition from a correlated input indistinguishable function family to a 1-$\mathcal{C}$-aRKA-wPRF secure function family. The 1-$\mathcal{C}$-aRKA-wPRF experiment runs as the general Φ-RKA with the exception that all the queries are related to a single random but public input r. Moreover, the set of key deriving function considered is restricted to a class $\mathcal{C}$ of polynomial size circuits $\mathcal{K} \to \mathcal{K}$

Definition 2.31 (1-$\mathcal{C}$-aRKA-wPRF security)**.** *Let $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be an efficiently computable function. f is* 1-$\mathcal{C}$-aRKA-wPRF *if, for all probabilistic polynomial time $\mathcal{A}$, there exists a negligible function* negl *such that:*

$$\left|\Pr\left[k \leftarrow_{\$} \mathcal{K}, r \leftarrow_{\$} \mathcal{D} : \mathcal{A}(r, \mathcal{O}^{r}_{f(RK(\cdot,k),\cdot)}) \to 1\right]\right.$$
$$\left. - \Pr\left[k \leftarrow_{\$} \mathcal{K}, r \leftarrow_{\$} \mathcal{D}, g \leftarrow_{\$} g \leftarrow_{\$} \mathrm{FF}(\mathcal{K},\mathcal{D},\mathcal{R}) : \mathcal{A}(r, \mathcal{O}^{r}_{g(RK(\cdot,k),\cdot)}) \to 1\right]\right| \leq \mathsf{negl}(\lambda)\,,$$

where $\mathcal{O}^{r}_{f(k,\cdot)}$ (resp. $\mathcal{O}^{r}_{g(k,\cdot)}$) takes as input a circuit $C_i \in \mathcal{C}$ and outputs $f(C_i(k), r)$ (resp. $g(C_i(k), r)$).

Input-Key Switching. The equivalence between correlated input pseudo randomness and related key security is obtain by switching the role of the key and the input. From any family of functions $f \in \mathrm{FF}(\mathcal{K},\mathcal{D},\mathcal{R})$ on can construct a family of function $\tilde{f}$ in $\mathrm{FF}(\mathcal{D},\mathcal{K},\mathcal{R})$. For any x in $\mathcal{D}$, $\tilde{f}(x,\cdot)$ is a function from $\mathcal{K}$ to $\mathcal{R}$ defined as $\tilde{f}(x,k) = f(k,x)$.

[10] The dependancy on the security parameter is hidden in the size of the key space. For more formalism, one can define a function family with a generation algorithm that, on input the security parameter, returns a keyspace as well as a domain and a range.

2.6 Zero-Knowledge Proofs

Zero-knowledge proofs are a powerful cryptographic tool, which goal may seem counter intuitive: proving the knowledge of an information without revealing nothing about it. The intuition about proof of knowledge and zero knowledge proofs first appeared in [GMR89] and was further formalised in [BG93]. Consider an NP relation $\mathcal{R}$, *i.e.* given a witness w and an input x, $\mathcal{R}(x,w) = 1$ can be decided in polynomial time. Let $\mathcal{L}$ be the language associated to $\mathcal{R}$, $\mathcal{L} = \{x | \exists w \text{ such that } \mathcal{R}(x,w) = 1\}$. We note $\mathcal{R}_{\mathcal{L}}(x)$ the set of witness for x for the relation associated to the language $\mathcal{L}$. An interactive proof system is a two party protocol between a prover P and a verifier V. Given a public input x and a relation $\mathcal{R}$, the former aims at convincing the latter that there exists a witness w such that $\mathcal{R}(x,w) = 1$.

Definition 2.32 (Proof system)**.** *Let (P,V) be a pair of interactive algorithms such that, on input a public statement x, P and V interacts such that, at the end of interactions, V accepts the proof or not. Let $tr_{P,V}(x)$ represent the transcript of the interaction between P and V, together with the final decision of V. We say that $tr_{P,V}(x)$ is an accepting conversation ($tr_{P,V}(x) \in accept$) if it corresponds to an interaction ending with V accepting the proof. Else, $tr_{P,V}(x)$ is an rejecting conversation ($tr_{P,V}(x) \in reject$). The pair (P,V) defines an interactive proof system if it verifies:*

- *Completeness. For every $x \in \mathcal{L}$, $\Pr[tr_{P,V}(x) \in accept] = 1$ (if the verifier and the prover are honests and behave correctly, the verifier will always accept).*
- *Soundness. There exists a negligible function* negl *such that, for every $x \in \{0,1\}^{\lambda} \setminus \mathcal{L}$ and every malicious prover, P^* $\Pr[tr_{P^*,V}(x) \in accept] \leq \mathsf{negl}(\lambda)$ (the verifier shall not accept in an execution with a cheating prover except with negligible probability).*

Zero-Knowledge proofs. A proof system is said to be zero-knowledge if it reveals nothing beyond the validity of the assertion of the prover. Formally, a proof system (P,V) for a language $\mathcal{L}$ is said to be perfect/statistical/computational zero-knowledge if, for every potentially malicious verifier V^*, there exists a Simulator $\mathcal{S}im$ running in polynomial time on input x and a black box access to the prover P such that:

$\{tr_{P,V}(x)\}_{x \in \mathcal{L}}$ (the distribution of transcripts of a real execution of the protocol between P and V^*), and

$\mathcal{S}im(V^*, x)_{x \in \mathcal{L}}$ (the distribution of transcripts created by the simulator)

are identical/statistically/computationally indistinguishable.

Proof of Knowledge. A Proof of Knowledge (PoK) allows the prover to convince the verifier that he knows a witness w, without revealing it. The soundness property is replaced by the existence of a polynomial time extractor Ext, defined as follows:

Definition 2.33 (Proof of knowledge)**.** *Let $\mathcal{R}$ be a relation. Let κ be a function, $\kappa : \{0,1\}^* \to [0,1]$. Let (P,V) be a pair of interactive algorithms such that, on input a public statement x, P and V interacts such that, at the end of interactions, V accepts the proof or not. (P,V) is a proof of knowledge for the relation $\mathcal{R}$ with knowledge error κ if it is complete and there exists a polynomial time extractor* Ext *such that, for any malicious prover P^*, on input a public statement x such that*

$p(x) = \Pr[tr_{P^*,V}(x) \in accept] > \kappa(x)$, and given a black box access to P^*, there exists a constant c such that Ext *outputs a witness $w \in \mathcal{R}_{\mathcal{L}}(x)$ with an expected number of step bounded by:*

$$\frac{|x|^c}{p(x) - \kappa(x)}.$$

Following the notation of [CS97], we write $\mathsf{PK}\{w_1, \ldots, w_s : \mathbb{R}(w_1, \ldots, w_s, x_1 \ldots, x_t) = 1\}$ to denote the proof of knowledge of the secret witnesses $w_1, \ldots, w_s$ that satisfy the relation $\mathcal{R}$ with the public values $x_1, \ldots, x_t$.

Argument of Knowledge. A ZK Argument is a ZKPoK with computational soundness instead of statistical soundness. Statistical soundness guaranties security even against an unbounded cheating prover, whereas computational soundness only holds against a polynomial time cheating P^*. In the former case, a proof system always leads to a ZKPoK as an unbounded P^* will be able to compute a witness.

2.6.1 Sigma protocols

A Sigma protocol (Σ protocol) is a three moves protocol in which a prover convinces a verifier that he knows a witness w for a relation $\mathbb{R}(x, w)$. The prover first sends a commitment a, then the verifier challenges him with a value c. Finally, the prover answers the challenge with a value t. The verifier accepts if the values x, a, c, t satisfy an efficiently computable predicate Pre on the statement to be proven. A Σ protocol moreover shall respect the following properties :

- *Completeness*: if the prover knows a witness w and the prover and the verifier follow the protocol, then the verifier always accepts.
- *s-Special soundness*: a Σ protocol has s-special soundness σ if, given s distinct transcripts $(a, e_i, t_i)_{i \in [1..n]}$ of the protocol, a witness w such that $\mathbb{R}(x, w)$ can be efficiently computed.
- *Special honest-verifier Zero-Knowledge*: there exists a PPT Simulator $\mathcal{S}im$ which, given any public statement x and a random challenge e, can return a transcript (a, e, c) with a probability distribution indistinguishable from the distribution of a real execution of the protocol with an honest verifier. If the verifier, denoted V^*, is not honest and tries to obtain information by cheating, then the protocol is said to be full Zero-Knowledge.

A Σ-protocol defines an honest-verifier ZKPoK with knowledge error $|C|$, as proven in [Dam10], where $|C|$ is the cardinal of all possible challenges. Hence the challenge space shall be large enough for the knowledge error to be negligible. The honest verifier restriction implies that the verifier is supposed to generate the challenge randomly. Otherwise, the simulation is not possible anymore. The s-Special soundness considers soundness once given s distinct transcripts. A malicious prover shall not be able to produce more than $s - 1$ valid proofs. When proving the classical soundness from the s-special soundness, the extractor uses rewinding to obtain the desired number of distinct transcripts. The main point is that the extractor shall run in polynomial time. An efficient rewinding is given in [Dam10].

2.6.2 The Fiat Shamir Transform

The Fiat Shamir heuristic was introduced by Fiat and Shamir in [FS87] The original goal was to design an efficient signature scheme from an interactive identification scheme. We focus our example on Sigma protocols. As shown in Figure 2.8, given a traditional Sigma protocol Π, the Fiat Shamir transform removes the interaction with the verifier who traditionally sends a challenge. In the protocol FS_Π, the prover generates the challenge itself by calling a $h(x, a)$ were a is the commitment value and x the public statement.

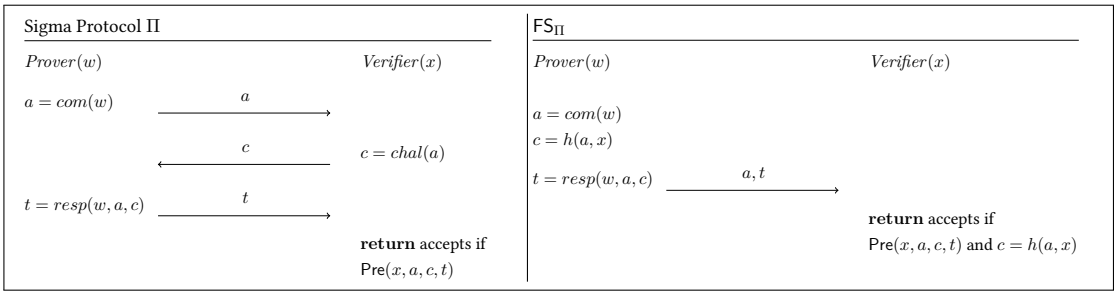


Figure 2.8 – The Fiat Shamir transformation of a Σ protocol.

The Fiat Shamir transform was proven secure in the ROM for signatures by Pointcheval and Stern in [PS96b] (extended in[PS96a]). The proof is based on a result known as the forking lemma. This result relies on probability properties and ensures that, given a sound interactive three move authentication protocol Π, then the Fiat Shamir transform of this protocol, FS_Π is a secure signature. In the context of Sigma protocols, the reduction consists in showing that an adversary who can produce a fake transcript for the non interactive FS_Π induces an adversary that produces two transcripts and then breaks the 2 special soundness of the original protocol Π, which is contradictory as Π was supposed sound. The zero-knowledge property does not come so clearly. It requires a property called programmability. The idea is that the simulator for FS_Π can call the simulator for the original protocol so as to obtain a valid simulation (a, e, t) (valid is to be understood as indistinguishable from the distribution of real transcripts). Then if he can program his oracle such that $H(a) = e$, the simulation is a valid simulation for FS_Π. A non negligible advantage is that it is sufficient to have honest verifier zero knowledge for the original interactive protocol. Despite the limits of the ROM underlined in subsection 2.5.1, the Fiat Shamir transform is still widely used as it enables practical non interactive protocol with nearly no overhead.

2.6.3 ZK proofs on circuits

Sigma protocols provide efficient proof system for algebraic statements. But not all statements can be efficiently expressed as an algebraic relation. A circuit is a computation model, defined as an acyclic directed graph composed of wires and gates. It can be algebraic, running over a field $\mathbb{F}$, with addition and multiplication gates, or boolean, running over $\mathbb{Z}_2$ with XOR and AND gates. We focus here on proof systems relative to a circuit execution: given a circuit C, proving the knowledge of a witness w such that $C(w, x) = 1$.

Garbled Circuits. In 2013, Jawurek, Kershbaum and Orlandi adapt in [JKO13] the 2 party computation garbling scheme of Yao ([Yao82]) to the zero-knowledge. Basically, the idea is that the Verifier

computes an encrypted version of the circuit. The prover asks for the garbled values corresponding to its input secret, without revealing it, using an oblivious transfer protocol, such that he can evaluate the encrypted output and commit to it. Some more communications are needed for the prover to be sure that the circuit was garbled honestly, then the verifier can decrypt the output and check the validity of the result. This proposition introduced a great improvement for circuit based zero-knowledge. However, one main drawback is that it is inherently interactive. The original publication announces, for proving the knowledge of a pre-image of SHA 256, a proving time around 5,7s, a verification time around 4,4s, for a communication complexity of 3MB.

SNARKs SNARKs are Succinct Non interactive ARgument of Knowledge. Succinctness ensures that the proof size remains at most logarithmic in the size of the public instance. This property was first put forward for argument systems by Kilian ([Kil92]) and Micali ([Mic94]) and then extended more recently to arguments of knowledge (see for instance [BCC+14]). SNARKs are settled in the common reference string (CRS) model. This model supposes that the prover and the verifier share a common uniform public random string. This requires the existence of a trusted third party that distributes such an input. This constraint however avoids resorting to the ROM. General SNARKs can be constructed from two representative classes of NP problem: probabilistically checkable proofs (PCP) or span programs.

SNARK from PCP. Given a language $\mathcal{L}$, a probabilistically checkable proof system for $\mathcal{L}$ consists in a prover and a probabilistic verifier. For any statement $x \in \mathcal{L}$, the prover produces a proof Π that the verifier can check with $r(n)$ random coins and accessing $q(n)$ bits of Π. The verifier shall accept all honest proofs with probability 1 and shall not be mistaken on a false proofs with probability more than $1/2$. Such a PCP is denoted $\mathsf{PCP}(r(n), q(n))$. It has been proven in [ALM+98] that $\mathsf{PCP}(\log n, 1) = NP$, which means any language in NP is associated to a proof system where the verifier can decide based on $\log n$ random coins and a single bit of the proof. Kilian instantiated an interactive succinct argument from PCP in [Kil92], however concretely inefficient. More recently, [BCC+14] proposed a SNARK built from PCP.

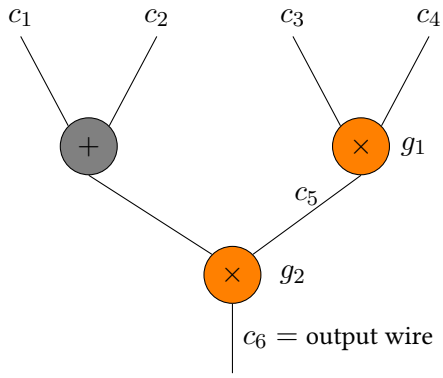
SNARK from SPAN programs. Most current efficient SNARK protocols are constructed from SPAN programs. Quadratic Arithmetic Programs (QAP, [PHGR13]) translate an arithmetic circuit into a set of polynomial constraints. They are a natural extension of Quadratic Span Programs introduced in [GGPR13] for boolean circuits.

Informally, each multiplication gate g_k, $k \in [1, |F|]$, (where $|F|$ denotes the complexity of the circuit representing the function F in terms of multiplication gates) is associated to an arbitrary field element r_{g_k}. The v_j's will encode the left input wire, such that $v_j(r_{g_k}) = 1$ if and only if the wire i is a left input wire to the gate g_k. Else $v_j(r_{g_k}) = 0$. Similarly, the w_j's encode the right input wire and the y_j's represent the output wire. See Figure 2.9 for an example. The idea is that those polynomial can encode the circuit. If we define a polynomial

$$\begin{aligned} p(x) &:= \left(v_0(x) + \sum_{i=0}^{m}(c_i v_i(x))\right) \cdot \left(w_0(x) + \sum_{i=0}^{m}(c_i w_i(x))\right) - \left(y_0(x) + \sum_{i=0}^{m}(c_i y_i(x))\right) \\ &:= V(x)W(x) - Y(x) \end{aligned}$$

a legal assignment is a set of coefficients $\{c_i\}_{i\in[0,m]}$ such that p vanishes on the roots $\{r_{g_k}\}_{k\in[1,|F|]}$ corresponding to the multiplication gates. In other words, if we set $t(x) = \prod_{i=0}^{m}(x - r_i)$, a legal assignment is a set of coefficients such that t divides p.

We recall the original definition from [PHGR13].



gate g_1:
left input: $v_3(r_{g_1}) = 1$, $v_j(r_{g_1}) = O$ for $j \neq 3$.
right input: $w_3(r_{g_1}) = 1$, $w_j(r_{g_1}) = O$ for $j \neq 3$.
output: $y_5(r_{g_1}) = 1$, $y_j(r_{g_1}) = O$ for $j \neq 5$.
gate g_2:
left input: $v_1(r_{g_2}) = 1$, $v_2(r_{g_2}) = 1$, $v_j(r_{g_2}) = O$ for $j \neq 1, 2$.
right input: $w_5(r_{g_2}) = 1$, $w_j(r_{g_2}) = O$ for $j \neq 5$.
output: $y_6(r_{g_2}) = 1$, $y_j(r_{g_2}) = O$ for $j \neq 6$.

$$t(x) = (x - r_{g_1})(x - r_{g_2})$$

Figure 2.9 – A toy example of QAP encoding an arithmetic circuit. The c_i's label the wire, r_{g_1}, r_{g_2} are the roots associated to the multiplication gates g_1 and g_2.

Definition 2.34 (QAP). *A Quadratic Arithmetic Program Q over the field $\mathbb{F}$ contains three sets of $m+1$ polynomials $V = \{v_i(x)\}, W = \{w_i(x)\}$ and $Y = \{y_i(x)\}, i \in 0, 1, \ldots, m$ and a target polynomial $t(x)$. Suppose F is an arithmetic function that takes as input n elements of $\mathbb{F}$ and outputs n_0 elements, for a total of $N = n + n_0$ I/O elements. Then, Q computes F if the following holds: $(c_1, \ldots, c_N) \in \mathbb{F}^N$ is a valid assignment of F's inputs and outputs, if and only if there exist coefficients $(c_{N+1}, \ldots, c_m)$ such that $t(x)$ divides $p(x)$, where:*

$$p(x) := \left(v_0(x) + \sum_{i=0}^{m}(c_i v_i(x))\right) \cdot \left(w_0(x) + \sum_{i=0}^{m}(c_i w_i(x))\right) - \left(y_0(x) + \sum_{i=0}^{m}(c_i y_i(x))\right).$$

In other words, there must exist some polynomial $h(x)$ such that $h(x) \cdot t(x) = p(x)$. The size of Q is m, and the degree is the degree of $t(x)$.

The idea for constructing SNARK from QAP is to leverage all the polynomial equations as exponents in a group that supports bilinear pairing. They achieve linear verifier time but only quasi linear prover time (generally $\mathcal{O}(|F| \log(|F|))$).

Considering the original zk-SNARK protocol Pinocchio described in [PHGR13], both the CRS and the proof computation require exponentiations in the group while the verification requires only one pairing computation. A proof of knowledge of a pre image for SHA-1 requires 12s for the CRS generation, 15.7s for the proof computation, around 10 ms. for the verification and the proof size is a constant of 8 group elements (288 Bytes).

For more details on SNARKs, both from PCP and QAP, we refer to [Nit19].

STARKs. Some more recent work focuse on removing the CRS constraint. A main reason is that, relying on a trustworthy party in a decentralized system - for instance crypto currencies, which are one the main application of SNARKs -, is challenging. Some implementations (for instance the original Zcash ([16]) choose to share the distribution via a multi party computation protocol, but this has a non negligible computational cost. A zero knowledge STARK (zkSTARK) still require some initial information to be shared between the prover and the verifier but make this information random and so, easier to deliver. Among the more relevant zkSTARK proposals, we identify Ligero ([AHIV17]), and its recent counterpart for boolean circuit BooLigero ([GSV21]), as well as Aurora ([BCR+19]) and STARK ([BBHR18]). The former presents an asymptotic proof size of $\mathcal{O}(\sqrt{N})$,

while the latter achieve $\mathcal{O}(\log^2 N)$. Those asymptotic results are not easily achieved concretely. For instance, STARK only shows efficiency for very large circuits (above 10^6 multiplication gates).

MPC in the Head. Ishai *et al.* introduced in [IKOS07] a new paradigm for ZK proofs, called MPC in the head. This solution reveals to be very competitive in terms of efficiency for ZK proofs performed on circuits, and does not require a CRS to be shared between the prover and the verifier. The idea is that the prover performs a virtual multi party computation (MPC) protocol and obtains several views. He commits to these views and opens only a sub-part of them, required by the verifier, to convince this party. We detail in the following paragraph a very intuitive version of MPC in the head.

ZKBoo. The seminal paper [GMO16] introducing ZKBoo proposes the first efficient ZK argument of a hash function computation. They improve the efficiency of previous MPC in the head arguments by introducing a function decomposition instead of more traditional MPC protocols. A (2,3)-decomposition is a set of functions that separates the evaluation of a function into three symmetric parts. With the requirement that, from any two parts, nothing is revealed about the third one, especially about the input.

Definition 2.35. *A (2,3)-decomposition for the function ϕ is the set of functions:* $D = (\mathsf{Share}, \mathsf{Output}_1, \mathsf{Output}_2, \mathsf{Output}_3, \mathsf{Rec}) \cup \mathcal{F}$ *such that:*

- Share *is a surjective function used to split the input x in three shares x_1, x_2, x_3;*
- *$\mathcal{F}$ is a finite family of efficiently computable functions described as $\{\phi_1^j, \phi_2^j, \phi_3^j\}_{j\in[1,N]}$;*
- $\mathsf{Output_i}$ *computes a value y_i called the output share;*
- Rec *computes the final value $y = \phi(x)$ from the three outputs shares y_1, y_2 and y_3.*

The evaluation of the function ϕ on input x through the decomposition, denoted $\Phi_\phi(x)$ produces three views w_i, $i \in \{1, 2, 3\}$, each composed of the input share x_i together with the output of the ϕ_i, j's and y_i. In the following, we will denote e, $e + 1$ $e + 2$ the three views for any $e \in \{1, 2, 3\}$. In particular, additions on indices are implicitly performed $\mod 3$.

Two properties are required from a decomposition: correctness and 2-privacy. The first means that the decomposition enables to correctly evaluate the function. The second guarantees that given any two views, one cannot learn the secret input x.

Definition 2.36. *Properties of a decomposition.*

- *Correctness : a (2,3)-decomposition $\mathcal{D}$ is said to be correct if $\Pr[\phi(x) = \Pi_\phi(x)] = 1$ for all $x \in X$. The probability is over the choice of the random tapes k_i.*
- *Privacy: a (2,3)-decomposition $\mathcal{D}$ is said to be 2-private if it is correct and, for all $e \in \{1, 2, 3\}$ there exists a simulator $\mathcal{S}im_e$, running in polynomial time, such that $(\{k_i, w_i\}_{i=e,e+1}, y_{e+2})$ and $\mathcal{S}_e(\phi, y)$ output the same probability distribution for all $x \in X$.*

The protocol ZKBoo, described in Figure 2.10 is a ZK argument/sigma protocol built from (2,3)-decomposition in the same way IKOS ([IKOS07]) did from MPC, as formalised in the Theorem 2.4, given in [GMO16]:

Proposition 2.4. *The ZKBoo protocol is a Σ-protocol for the relation $\mathcal{R}_\Phi$, with 3-special soundness.*

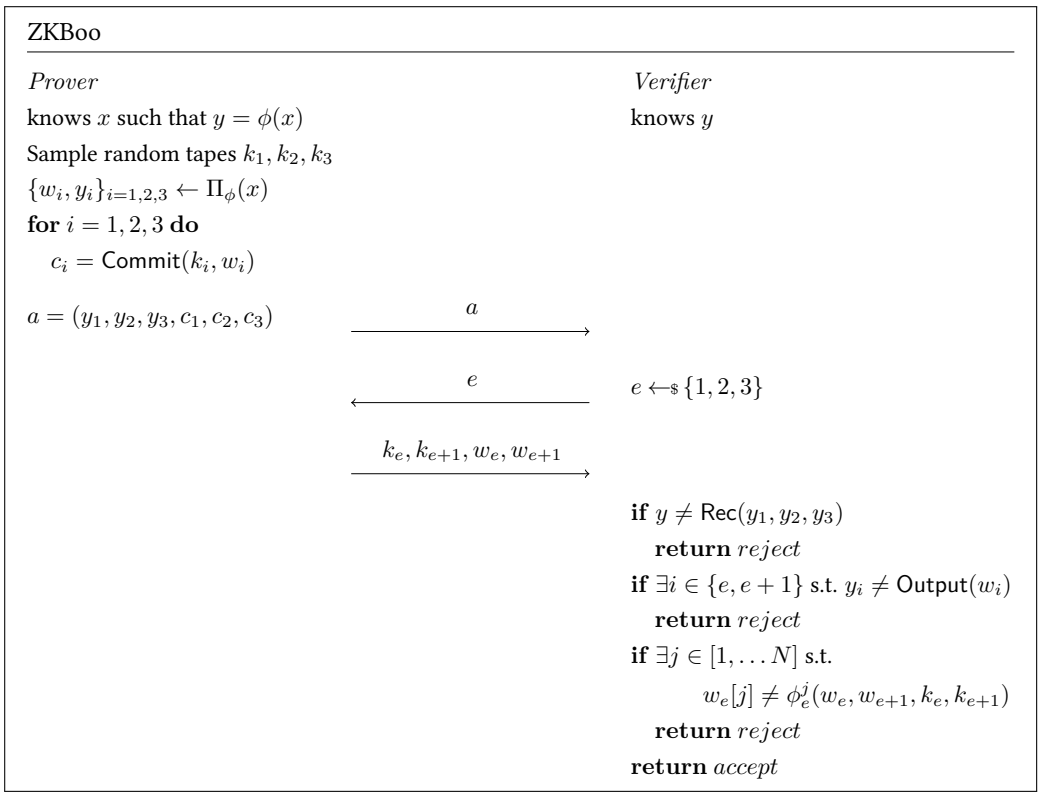


Figure 2.10 – The ZKBoo sigma protocol built from a (2,3)-decomposition of a function.

A soundness probability of $2^{-\sigma}$ (meaning the probability for a cheating $Prover$ not to be caught is less than $2^{-\sigma}$), is obtained by repeating the process $t = \sigma \cdot (\log_2 3 - 1)^{-1}$ times.

We finally give an overview of the practical instantiation of a (2,3) decomposition for any circuit (boolean or arithmetic) that evaluates a function $\Phi : R^k \to R^\ell$ where R is a ring. Each gate, except the multiplication gates, evaluates normally (or quiet, as addition by a constant is only performed in the first view), whereas the computation of the output wire c of a multiplication gate with input wires a and b is specifically split among the three views and randomized so as to obtain the 2-privacy property.

$$\begin{aligned} w_e[c] &= \phi_e^c(w_i[a,b], w_{e+1}[a,b]) \\ &= w_e[a] \cdot w_e[b] + w_i[a] \cdot w_{e+1}[b] + w_e[b] \cdot w_{e+1}[a] + R_e(c) - R_{e+1}(c), \end{aligned}$$

with $R_e(c)$ (respectively $R_{e+1}(c)$) being the randomness associated to view e (respectively $e+1$) for the gate c. The initial sharing is a simple additive secret sharing, where addition is taken on R and the output y_e consists in the value of the output wire in the view w_e. Then the recomposition Rec evaluates the addition of the three output shares: $\mathsf{Rec}(y_1, y_2, y_3) = y_1 + y_2 + y_3$, where the addition is taken on R. This description is proven to be a (2,3) decomposition, meaning that it satisfies the correctness and 2-privacy properties. Note that for their implementation of ZKBoo proofs on SHA-1 and SHA-256, the authors consider a circuit described on $\mathbb{Z}_2$ so additions are to be understood as XOR and multiplications as binary AND.

Efficiency. For $|c|$ the bit size of each commitment, $|r|$ the bit size of each random element r_e and $|k|$ the bit size of a random-tape k_e, one obtains, for t rounds, a proof size of $|proof| =$

$t \cdot [3(|y| + |c|) + 2 \cdot (|view| + |k| + |r|)]$. In [CDG+17] the authors propose an improvement of ZKBOO, called ZKBoo++. They avoid sending values or commitment that the *Verifier* is able to compute, once revealed the proof. They gain a factor two in the proof size at no extra computational cost. They obtain $|proof| = t \cdot [|c| + 2 \cdot |k| + 2/3 \cdot |x| + |view| + |chal|)]$ where $chal$ is the challenge computed from the commitments to the three views, in the Fiat Shamir transform. Focusing on the size of a view, it is easy to remark that, from an input share e, a *Verifier* can compute all gates of the view_e except the multiplication ones. For these, the *Verifier* needs values from the next view view_{e+1}, which can also be reduced to the multiplication gates output wires values. Hence the size of the view is fully determined by the number of multiplication gates in the circuit. The size of the proof is linear in the number of multiplication gates in the circuits.

PQ signature and further improvements. The efficiency of MPC in the head based protocol for circuit proof gave birth to a family of Post Quantum signature called PicNic, that was submitted to the NIST Post Quantum competition[11]. The improvement ZKBoo++ ([CDG+17]) was originally included in the first Post Quantum signature scheme PicNic. With the same motivation, the KKW protocol ([KKW18]) or the recent TurboIKOS [GHS+21] are also improved MPC in the head protocols. The former is included in PicNic2 and PicNic3 (the differences between version 2 and 3 are mainly optimisation in the implementation of KKW). It reaches better performance than the original ZKBoo++, while using preprocessing. The later reaches performance close to KKW but is implemented in alternative version of the PicNic scheme, which makes practical comparison more difficult.

2.6.4 Commitment schemes

In the context of zero-knowledge, the prover often needs to convince the verifier that he makes use of a certain fixed value, without revealing this value. This is the purpose of commitment : to bind to a value without leaking information about it.

Definition 2.37 (Commitment scheme)**.** *A non interactive commitment scheme is definied by three algorithms :*

- $\mathsf{Gen}(\lambda)$ *a generation algorithm that, on input a security parameter* λ*, outputs the public parameters* pp *for the scheme. They will be implicit input off the following algorithms.*
- $\mathsf{Com}(m, r)$ *on input a message* m *and some random coin* $r \in \mathbb{R}$*, outputs a commitment* c *and an opening information* d*.*
- $\mathsf{Open}(c, d)$ *on input a commitment* c *and an opening information* d*, returns* `accept` *or* `reject`*.*

On the one hand, a commitment c shall not reveal any information about the committed x; this is the *hiding* property. On the other hand, only the secret x shall produce a valid opening for c; this is the *binding* property. We provide below formal definitions for both those properties.

Definition 2.38 (Hiding)**.** *A commitment scheme is said to be perfectly (respectively statistically, computationally) hiding if for all* $x \neq x'$*, the following distributions:*

$$\{\mathsf{Com}(x, r)\}_{\mathcal{U}(\mathbb{R})} \text{ and } \{\mathsf{Com}(x', r)\}_{\mathcal{U}(\mathbb{R})}$$

are perfectly (resp. statistically, computationally) indistinguishable (where $\mathcal{U}(\mathbb{R})$ *is the uniform distribution over* $\mathbb{R}$*).*

[11] The PicNic scheme reached the third round as an alternate candidate. However, due to recent cryptanalysis that impacted multivariate candidates GeMSS and Rainbow, a fourth round opened to new candidate is actually under discussion.

Definition 2.39 (Computational Binding)**.** *A commitment scheme is said to be computationally binding if, for all probabilistic adversaries $\mathcal{A}$ running in polynomial time, there is a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\Pr\Big[pp \leftarrow \mathsf{Gen}(1^\lambda), (m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(pp) : m_0 \neq m_1 \wedge \mathsf{Com}(m_0, r_0) = \mathsf{Com}(m_1, r_1)\Big] \leq \mathsf{negl}(\lambda)\,.$$

where m_0, m_1 are valid message for the commitment scheme, $r_0, r_1 \in \mathbb{R}$.

A commitment scheme can not be both statistically hiding and statistically binding. Informally, the former requires the commitment to open at least to two different values (otherwise an unbounded prover that can test all the input to break the hiding property), but then, it is not binding anymore.

In addition, it is sometimes desirable that a third party (a simulator for instance) can cheat the binding and open the commitment to the value of its choice. Some commitment schemes offer this possibility, with the help of an auxiliary trapdoor value. Such schemes are said to be *equivocable.*

Definition 2.40 (Equivocality)**.** *A commitment scheme is said to be equivocable if there exists an alternative generation algorithm* Gen' *and an algorithm* Eval *such that:*

- *on input the security parameter,* $\mathsf{Gen}'(1^\lambda)$ *returns the public parameters pp and a trapdoor T;*
- *for any commitment $C = \mathsf{Com}(m, r)$, and any message m',* $\mathsf{Com}(m', \mathsf{Eval}(T, m', (C, d)) = C$.

Moreover, there exists an extractor algorithm $\mathsf{Ext}_{\mathsf{Com}}$ *that, given two distinct valid opening d_0, d_1 on a commitment C, can extract the trapdoor T.*

Perdersen commitment. The Pedersen commitment scheme [Ped92] is an equivocable scheme with unconditional hiding and computational binding. It is routinely used because it interacts nicely with linear relations. This scheme is defined as follows: let $\mathbb{G}$ be a cyclic group of prime order q, P a generator and $Q \in \mathbb{G}$ such that the discrete log of Q in base P is unknown. Then, $\mathsf{Com}(x) = xP + rQ$ where r is sampled at random in $\mathbb{Z}_q$. Let C_1, C_2 be commitments to values x_1, x_2. If $a, b \in \mathbb{Z}_q$ are public values, then one can efficiently prove the following : $PK\{x_1, x_2, r_1, r_2 : C_1 = x_1P + r_1Q \wedge C_2 = x_2P + r_2Q \wedge ax_1 + x_2 = b \mod q\}$. The trapdoor for equivocality is given by the discrete log of Q in base P.
We recall in Figure 2.11 the Σ-protocol to prove knowledge of the opening values of a Pedersen commitment C_x. That is, $\mathsf{PK}\{x, r : C_x = xP + rQ\}$. This proof is close to the discrete logarithm proof.

Proof of knowledge of a linear relation between two Pedersen commitments. We recall in Figure 2.12 the Σ-protocol to prove this statement: $\mathsf{PK}\{x_1, x_2, r_1, r_2 : C_1 = x_1P + r_1Q \wedge C_2 = x_2P + r_2Q \wedge ax_1 + x_2 = b \mod q\}$ and we give the associated proof.

Proof. Correctness follows by inspection.

Soundness. An extractor works as follows: given two transcripts with fixed values for d_1, d_2 but distinct challenges e_1, e_2 (opportunity given by the forking lemma), it can evaluate, as in the original Schnorr proof, the witnesses values x_1, x_2, r_1, r_2 and check that the linear relation is verified.

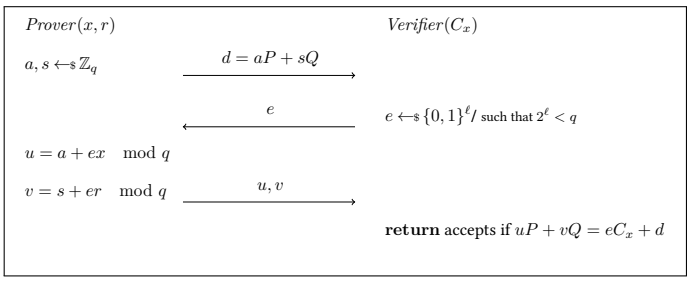


Figure 2.11 – Proof of knowledge of opening of Pedersen commitment

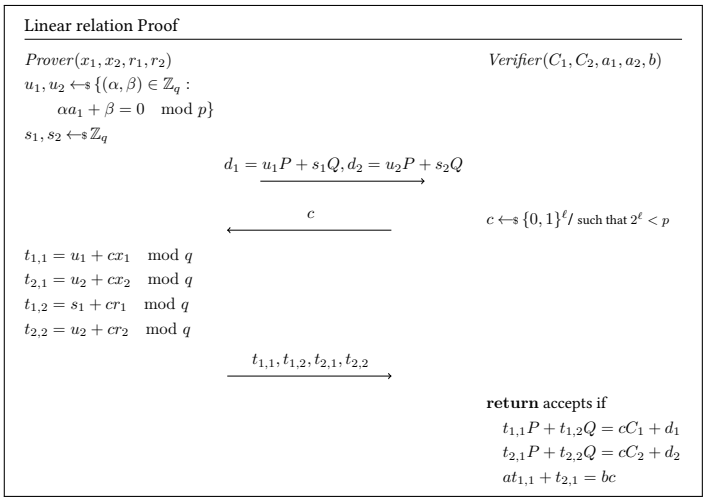


Figure 2.12 – Proof of knowledge of a linear relation between two Pedersen Commitments

Honest verifier zero knowledge. A simulator, knowing only the public values C_1, C_2, a, b can simulate a correct view as follow. As a first step he guesses what the challenge e will be. Then he samples a value $t_{1,1}$ and deduces $t_{2,1}$ from the relation $at_{1,1} + t_{2,1} = be$. Then he samples $t_{1,2}$ and $t_{2,2}$ and deduces d_1 and d_2. He sends them to the $Verifier$. He receives the challenge from the Verifier. If it is not the expected challenge then he aborts. Otherwise he answers with the pre calculated values. Values $d_1, d_2, t_{1,1}, t_{1,2}, t_{2,1}, t_{2,2}$ are as random as the original should be (they are sampled at random whereas the original ones are evaluated from at least on e random coefficient picked in the same space). The Verifier cannot distinguish the two distributions.
It is important to note that a valid simulation can be obtained even if the linear relation is not verified. Suppose b is not a fixed public value but that the $Prover$ samples b at random. As he does not know x_1, x_2, he can not predict whether the relation $ax_1 + x_2 = b$ holds. However the above simulation remains valid. □

2.7 Verifiable encryption

Verifiable encryption aims at convincing a verifier that an encrypted data satisfies some properties without leaking any information about the data itself. In such 2-party protocol, a prover and a verifier share in a common input string a public key encryption scheme Enc, a public key pk for

Enc, and a public value y. At the end, the verifier either accepts and obtains the encryption of a secret value x under pk such that x and y verify some relation $\mathcal{R}$ or rejects. It is worth noticing that the prover does not need the secret key sk, that usually belongs to a third party. Verifiable encryption often appears in the domain of anonymous credentials, fair exchange signatures, or verifiable secret sharing [Sta96]. In [CD00], Camenish and Damgård describe how to provide a proof that an encrypted value is a valid signature, using any semantically secure encryption scheme. The idea is to take advantage of the Σ-protocol for a relation $\mathcal{R}(x, y)$, to prove that an encrypted value is the witness x for this relation.

2.7.1 A formal definition

We recall here the formal definition of verifiable encryption as detailed in [CD00]. Let (KeyGen, Enc, Dec) be a probabilistic public key encryption scheme and $(pk, sk) = \mathsf{KeyGen}(1^\lambda)$ a valid key pair. The verifiable encryption mechanism, attached to an encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, to the binary relation $\mathcal{R}$ and to the associated language $L_\mathcal{R} = \{(x, w) : \mathcal{R}(x, w) = 1\}$, is defined as a two-party protocol Π between a prover P (who encrypts the data) and a couple composed of a verifier V on the one hand and a recovery algorithm Rec on the other hand. The protocol Π takes as public parameters a valid public key pk, a statement x and a security parameter λ. Let $V_P(pk, x, \lambda)$ denote the final output of V interacting with P on input (pk, x, λ). The recovery algorithm takes as input the secret key sk and $V_P(pk, x, \lambda)$.

Definition 2.41 (Secure Verifiable encryption)**.** *The couple protocol/recovery algorithm described above is a secure verifiable encryption scheme if the following holds:*

- *completeness: if P and V are honest, then $V_P(pk, x, \lambda) \neq \perp$ for all (pk, sk) valid key pair for the subsequent encryption scheme and $x \in L_\mathcal{R}$;*
- *validity: for all PPT malicious prover P^*, all valid key pairs (sk, pk), $\Pr[\mathcal{R}(x, \mathsf{Rec}(sk, V_{P^*}(pk, x, \lambda))] \neq 1$ and $V_{P^*}(pk, x, \lambda) \neq \perp)$ is negligible;*
- *computational Zero-Knowledge: for every unbounded malicious verifier V^*, there exists an expected poly-time Simulator $\mathcal{S}im_{V^*}$ with black-box access to V^* such that for all distinguisher A, all positive polynomial $p(\cdot)$, all $x \in \mathcal{L}$ and all sufficiently large λ we have:*

$$\Pr[A(pk, x, \alpha_i] = i : \quad (pk, sk) = \mathsf{KeyGen}(1^\lambda), \alpha_0 = V_P(pk, x, \lambda), \\ \alpha_1 = \mathcal{S}im_{V^*}(pk, x, \lambda), i \in \{0, 1\}) \leq \tfrac{1}{2} + \tfrac{1}{p(\lambda)}.$$

Informally, validity ensures that a malicious prover $\mathcal{P}^*$ will always be caught, except with negligible probability, because the recovery algorithm shall be able to compute a witness. The recovery success guaranties that the decryption will be correct. In the soundness property of ZKPoK, one needs a third party, the extractor, to unmask a cheating prover. We also note that the revelation process "kills" the Zero-Knowledge feature of the proof. Verifiable encryption follows the same rules, except that the third party - the recovery algorithm - needs an additional ingredient : the secret key sk. The verifier is not supposed to be honest here.

We denote by $\mathsf{VerifEnc}_{\mathsf{Enc},pk}(m : r)$ the encryption of a message m (using randomness r) under the public key pk with the encryption scheme Enc and the associated proof *i.e.* the output of the prover. We omit the randomness r when it is not necessary to explicitly mention it.

2.7.2 The Camenish-Damgård verifiable encryption scheme

In Figure 2.13, we describe the verifiable encryption solution given in [CD00], adapted to the Σ-protocol dedicated to a Pedersen commitment as defined in section 2.6.4

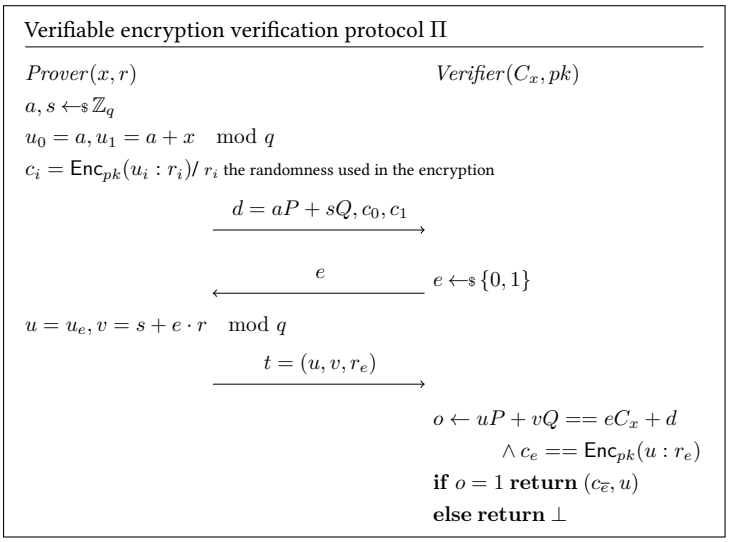


Figure 2.13 – A simple version of verifiable encryption scheme. The *Verifier* knows a public commitment to x, C_x and a public key pk. The *Prover* proves knowledge of the encrypted message x and of the random r used in the commitment C_x.

As for any cut-and-choose protocols, the probability that a cheating prover wins is $\frac{1}{2}$ for one round. One has to repeat the protocol σ times to obtain a cheating probability (a validity error) of $2^{-\sigma}$. The protocol described in Figure 2.13 can be optimized by gathering all rounds in a single one as described in the original paper, dropping to $\mathcal{O}(\log(\sigma))$ the number of encryptions to store.

About the verifiable encryption in [CPZ20]. In this paper, the authors describe a symmetric encryption scheme, inspired by the original ElGamal scheme, on which one can prove that the encrypted data is an attribute for which a specific anonymous algebraic credential was previously delivered. To encrypt a scalar message y, they call an $\mathsf{EncodeTo}\mathbb{G} : \{0,1\}^{\ell} \to \mathbb{G}$, an injective and easily reversible encoding to the group $\mathbb{G}$. The key in this scheme is that the prover can give a commitment C_e on the encoded value $\mathsf{EncodeTo}\mathbb{G}(y)$ together with a proof that the value committed in C_e is the encoded version of the value committed in another commitment C_y. And this is possible because the verifier plays an active part in the creation of these commitments (he delivers the original credentials and public values corresponding to secrets he owns. The *Prover* then uses those credentials and parameters to provide the desired proof.) Unfortunately, in our own case, the *Verifier* plays no part in the protocol apart verifying and transmitting values. Another drawback is that it requires a group that supports an injective encoding $\mathsf{EncodeTo}\mathbb{G}$ that comes with a zero-knowledge proof $PK\{(m, y) : y = \mathsf{EncodeTo}\mathbb{G}(m)\}$. They use the Ristretto255 group ([HVLT19]), a prime order group built on top of the Bernstein elliptic curve Curve25519 ([Ber06]).

3 Ratcheted Key Exchanges

Secure Instant Messaging solutions aim at offering to their users a quiet and private room to communicate. A non negligible number of solutions have emerged through the last decade, among them WhatsApp (1.5 billion of users, for 60 billions of messages sent each day [1]), Facebook Messenger with its optional secret conversation mode (1,3 billion of users [2]), Wire, Viber, Google Allo and Signal. Much more names could be mentioned here (Telegram, Threema, ...) but a common denominator of the solutions cited above is that they settle their security on a solution called the Double Ratchet algorithm. After being released officially by Signal's team in 2016, this algorithm has been widely studied and has led to the formalisation of theoretical primitive baptised Ratcheted Key Exchanges (RKE). One of our goal in this Chapter, is to walk along the way that goes from practical protocols (OTR then Signal) to a complex cryptographic primitive, which associated expected security is not well defined yet.

Contents

[1] http://techcrunch.com - Facebook Q4 2017 earnings announcement
[2] www.socialmediatoday.com/news Facebook Messenger by the numbers 2019

Whenever studying the history of a technological advance, setting a starting point is nearly impossible. Each small step is based on earlier developments and so, one may go back far in the past to unfold the whole story. Hence, we must decide of a subjective beginning. Ours will be 2004 and the publication of the protocol Off-the Record [BGB04]. Other protocols already dealt with the security of communication before: TLS (back then called SSL, *cf.* section 2.4.3), S/MiME ([Ram99] for the version 3 of 1999, [RST19] for the current version), PGP ([Zim95]), or Trillian [21] are the most commonly cited. However, the first was dedicated to secure connections between clients and servers, S/MIME and PGP were mostly designed for emailing and the last one, (back then, as it still exists and has evolved since) ensured secrecy without authentication. On a more academic level, the messaging solution based on puncturable encryption proposed in [GM15] (inspired by [CHK03]), is also interested in the security of messaging, but only explores the PFS feature. The systematization of Knowledge of Unger *et al.* ([UDB+15]) provides a good survey of the existing solutions available in 2015, and we will regularly refer to their work. For now, we start with Off-the-Record (OTR in the following lines), as it contains the main ingredients that will be needed in the more recent Signal, on which we elaborate our first contribution.

3.1 OTR and Signal : the practical protocols

3.1.1 Off-the-Record: the birth

The name of this protocol gives a good intuition of the intended goal we described earlier: enable private conversations, that would be held without anyone around to record it. Three key notions are associated: secrecy, PFS and deniability. The first notion relates to end-to-end authenticated encryption. Messages are encrypted on Alice's device and should be decrypted by Bob only. Deniability says that no one should be able to tell that a particular message was sent by Alice to Bob. Note that this is different from anonymity, where no one should even know that Alice spoke to Bob. Once again, the idea is to keep the privacy one enjoys in the street: without a recording of the conversation (or a witness who heard everything), then Bob cannot prove that Alice made specific comments: it is word against word only.

OTR introduces two key ingredients: firstly, the use of short-lived session keys, to provide secrecy. Secondly, a Mac based chaining system to provide authentication of the data without using a signature scheme, which is, by nature, non deniable. The short-lived key computation is based on a classical Diffie-Hellman protocol, that we recall here: let $\mathbb{G}$ be a cyclic group of prime order q, g a generator for this group. Alice and Bob can compute a shared secret as follows: Alice samples $a \leftarrow_{\$} \mathbb{Z}_q$, sends the public g^a to Bob. Bob samples $b \leftarrow_{\$} \mathbb{Z}_q$, sends the public value g^b to Alice. Then both of them can compute a common secret $g^{ab} = g^{ba}$ that we later denote $\mathsf{DH}(g^a, g^b)$. The most basic re keying system would renew this whole protocol regularly. However, in order to save computation, one can recycle an exponent, and only half refresh the common secret as described in Figure 3.1.

This refreshing procedure is called ratcheting: once you have moved from $k_{i,i}$ to $k_{i+1,i}$ (respectively from $k_{i,i-1}$ to $k_{i,i}$), no step backward is possible, given CDH is a hard problem. The ratchet procedure also extends the notion of PFS. No long term key are at stake in the refreshing procedure described in Figure 3.1. However, PFS can be understood relatively to the current state: accessing b_i does not harm the security of key computed with b_j, $j \in [\![0; i-1]\!]$.

From this regularly ratcheted secret are derived two keys, one for data encryption, the other is a Mac key. Each new Diffie-Hellman share's integrity is protected under the previous Mac key, composing a chaining system, that bounds each message to the origin. However, this origin has

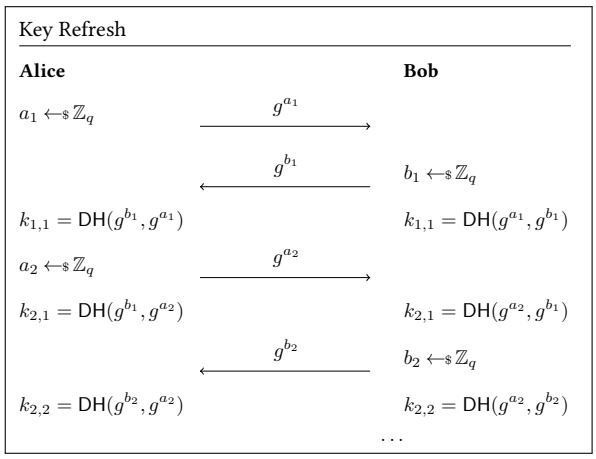


Figure 3.1 – The OTR Diffie Hellman based key refreshing mechanism, in its simplest form.

to be formally authenticated: Alice and Bob need to ensure they start on a good basis to rely on the chaining system. With this system, Alice encrypts her message and authenticate her new public Diffie-Hellman share under the keys that were computed at the previous round, as shown in Figure 3.2. OTR proposes a signature based authentication on the first two moves of the protocol. DiRaimondo *et al.* examine in [DGK05] the need for a strong authenticated key exchange to settle the authentication chain.

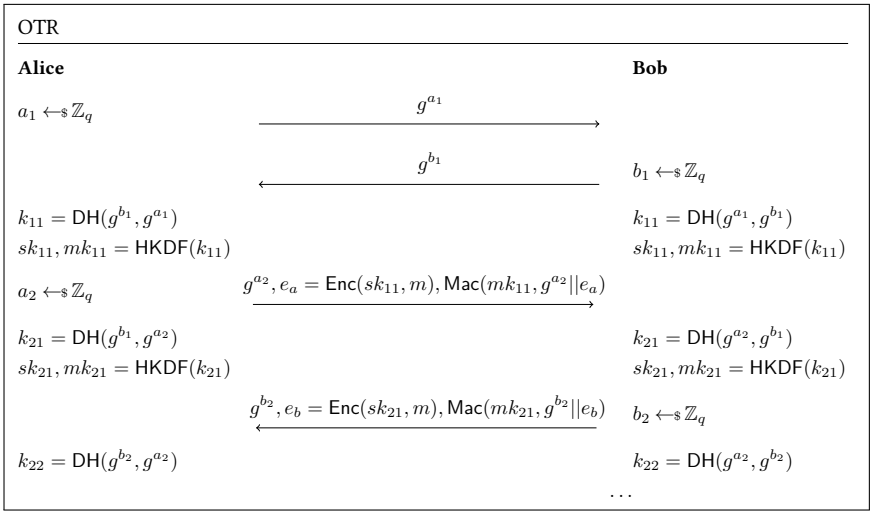


Figure 3.2 – Sending and authenticating messages in OTR.

We have now the main ingredients that will compose a ratcheted key exchange: an initial authenticated key exchange, a chaining authentication for deniability, and a regular update of the encryption keys: the ratchet itself. But OTR was designed for instant messaging on computers, that was, back then, inherently interactive. If the advent of smartphones saw the popularity of instant messaging applications massively raise, it also invalidated the paradigm that both participants in the conversation are on line at the same time. Secure messaging protocols have to move forward:

- The initial key exchange has to be non interactive;
- The authentication chain has to be adapted: as designed in Figure 3.2, Alice can not use her ratchet update as soon as she computes it. She has to wait for Bob to acknowledge it before. This is not suitable for the asynchronous setting.
- The "ping-pong" format of the ratchet,where the key evolves only when receiving a message, is not sufficient: as instant messaging session can now last for weeks or even years, if Bob is off line for a long time then no refreshing happens.

3.1.2 Signal: the confirmation

The Signal secure messaging application (previously named TextSecure) aims at proposing an asynchronous "end-to-end secure" messaging solution. It is based on two cryptographic protocols: the Double Ratchet algorithm ([MP16a]), that controls the refreshing mechanism of the session keys, and X3DH ([MP16b]), the initial asynchronous authenticated key exchange. Both protocols are often put together and identified as Signal[3]. Whatever the terms, we will focus in this manuscript on the cryptographic part of the application, that occurs on the user's device[4]. We first describe the protocol, then investigate on the first complete security analysis, provided by Cohn-Gordon *et al.* ([CCD+17]).

The initial key exchange. We recall the protocol X3DH in Figure 3.3. The main idea to reach asynchronism, is to let any participant who wishes to be contacted (here Bob) pre-publish some public keys on a dedicated server (Signal's server for instance):

- a long term public identity key pk_B. The first time Alice wants to communicate with Bob, she should check the validity of this long term key, using an alternative channel, for instance, by comparing a fingerprint value via a phone call or via a QR Code. Note that this mechanism does not prevent a malicious participant to register Bob's public key as its own, to perform an Unknown KeyShare Attack (UKS)[5], *cf.* section 2.4.1;
- a mid-term public key $prepk_B$. This key can be used for several initial key agreement, with different peers. However, it shall be renewed regularly. This frequency of renewal is decided at an application level. This prekey is signed (the public signing key can be thought of as part as the identity public key described above). This does not neither prevent UKS attacks. According to the specification [MP16b], this signature shall prevent a malicious user to register trapped prekeys on behalf of Bob, that would cancel the PFS of X3DH;
- a bunch of ephemeral pre-keys $epk_{B,j}$ for $j = 1, \ldots, M$. Each key is used only once. The last key of this set is called the last resort key. Its use is a warning that the set of ephemeral key should be renewed. If no one-time key is available, the last component of the key derivation secret (in cyan in Figure 3.3) is omitted.

On reusing mid-term keys. The reuse of ephemerals in Diffie Hellman based key agreements is studied in [MU10]. The authors show how this practice can lead to small subgroup attack. Consider

[3] Some time before the name Signal spreads in the security community, the protocol - either the ratchet only or together with the initial key exchange - was also identified as the Axolotl protocol.

[4] There is another layer of encryption and authentication between the Signal server and the Google Cloud Messaging system that distributes the message. We will not discuss it here.

[5] Moreover, this supposes the participants to be actively acting for the security of their communication. We extend on the peculiar role of the server in subsection 3.2.3

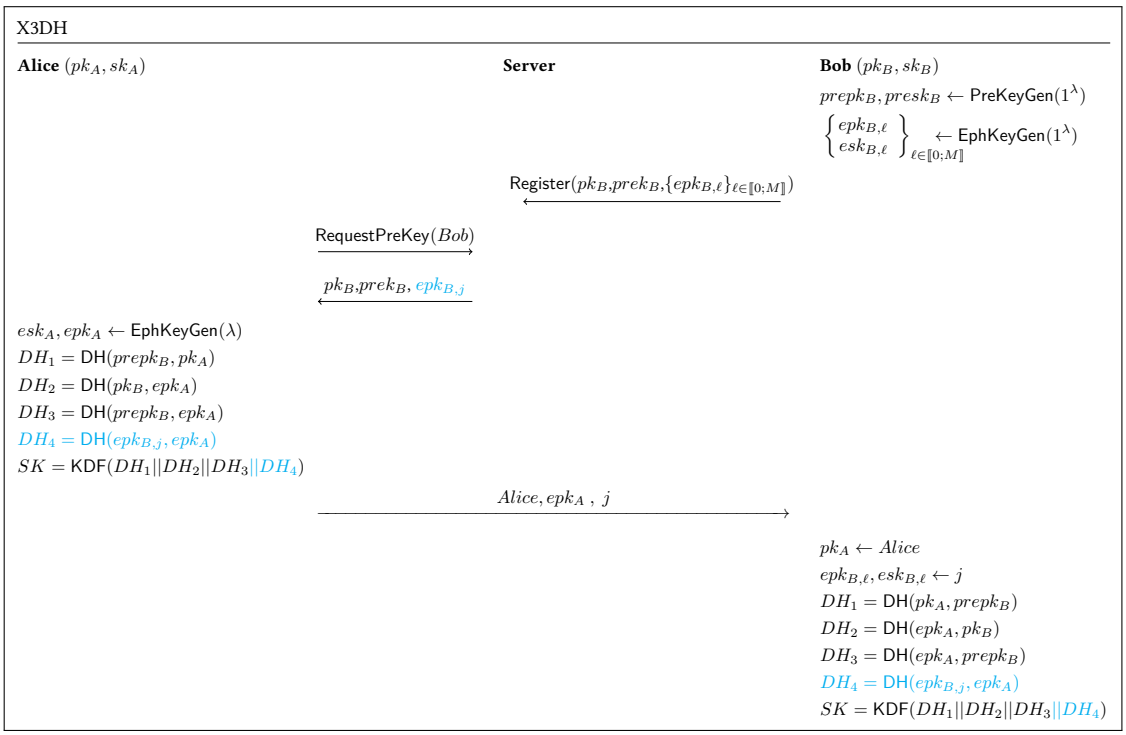


Figure 3.3 – The X3DH asynchronous authenticated key exchange. The elements written in cyan are optional. If no one-time key $epk_{B,j}$ is available, the protocol is performed with only the three first DH computations.

a Diffie Hellman based key exchange that takes place on a group $\mathbb{G}$ of prime order q. The adversary considers a group $\mathbb{G}'$ such that the elements in $\mathbb{G}'$ are encoded as the element in $\mathbb{G}$ and such that an implementation of the multiplication in $\mathbb{G}$ (addition for an additive group) is valid for $\mathbb{G}'$. Moreover, $\mathbb{G}'$ admits several elements g'_i of distinct low orders t_i (t_i shall be small enough for $\mathcal{A}$ to perform a brute force attack, $t_i \leq 160$). Then $\mathcal{A}$ registers one or several low order elements $pk_{B,i} = g_i'^{b_i}, b_i \leftarrow\!\!\$\, \mathbb{Z}_{t_i}$ as its own public key. If the key exchange enables Alice to reuse her keys a, g^a, $\mathcal{A}$ can brute force several values $a \bmod t_i$. The Chinese remainder theorem then gives a. One solution to prevent this attack is that Alice always check that Bob's public key is a valid element in $\mathbb{G}$, not equal to 1.

The Double Ratchet part. Once Alice and Bob share (virtually, as Bob may not have received the message yet) a common secret K, Alice can initiate the Double Ratchet algorithm to derive different keys. The Double Ratchet associates the asymmetric Diffie Hellman based ratchet of OTR, described in subsection 3.1.1, with another evolution: the short lived session key obtained from the asymmetric step is itself continuously updated with each message sent by Alice. This second derivation does not require any randomness from Bob, and can be done independently on both side, hence its name: symmetric ratchet. We list the keys and their function and give a schematic vision of the key schedule of Signal in Figure 3.4. Alice and Bob store and update the following keys:

- a ratchet key pair rsk_A, rpk_A: they are the ephemeral keys for the current asymmetric ratchet. We index the ratchet keys with a counter: $rsk_{A,i}, rpk_{A,i}$. As Alice and Bob alternatively generate new ratchet key, there can be at most a difference of one between their counters (the

initiator starts).

- a root key rk which takes charge of the authentication chaining system. This replaces the Mac chaining in OTR. The next epoch root key derivation depends on the current root key, this is the chaining. This way, the keys derived from this stage will be authenticated by Bob as coming from Alice by Bob as soon as he derives them. Hence they can be used once computed. The computation of the root key also depends on the asymmetric Diffie Hellman value. Hence we index it with a counter that evolves with each ratchet step. As Alice and Bob ratcheting counter can only differ by one, we obtain either $rk_{i,i-1}$ or $rk_{i,i}$. The common secret resulting from X3DH is the initial root key and called the master secret key MSK.

- a chain key ck: this is the starting point of the symmetric ratchet, from which are derived the message keys. An initial chain key is derived together with the root key as follows: $rk_{i,i}$, $ck_{i,i-1:0} = \mathsf{KDF}_r(\mathsf{DH}(rpk_{a,i}, rpk_{b,i-1}), 2\lambda, rk_{i,i-1}, cst_r)$ (respectively $rk_{i+1,i}$, $ck_{i,i:0} = \mathsf{KDF}_r(\mathsf{DH}(rpk_{a,i}, rpk_{b,i}), 2\lambda, rk_{i,i}, cst_r)$), where KDF_r corresponds to the HKDF key derivation function (*cf.* subsection 2.3.5). Then, a new chain key is derived with each message: $ck_{i,j:k+1} = \mathsf{HMAC}(ck_{i,j:k}, 0)$ ($j = i - 1$ or $j = i$). Note that the initiator of the session (here Alice) is always one index beyond the receiver. Hence, on Alice's side, keys indexed with $i + 1, i$ are computed when she wants to send messages: this initiates a sender chain. Keys indexed with i, i are computed on receiving a response from B and initiates a receiving chain. This is the contrary on Bob's side.

- a message key mk: this is the application key that encrypts the data exchanged between Alice and Bob. It is derived from the chain key by applying successively HMAC, then HKDF (without salt): $mk_{i,j:k} = \mathsf{HKDF}(\mathsf{HMAC}(ck_{i,j:k}, 1), cst_1)$ where cst_1 is a constant determined at the implementation level.

In [CCD+17], the author gather both the HMAC and HMAC+HKDF derivations of the chain key in a single global function KDF_m such that $\mathsf{KDF}_m(ck_{i,j:k}, cst_1) = ck_{i,ij,k+1}, mk_{i,j:k}$. This choice is on purpose to facilitate the exposure of there proof. As it also makes the global key schedule clearer, we adopt this notation in Figure 3.4.

The initial ratchet. Following the specification ([MP16a]), once Alice has computed the initial root key (the master secret), she immediately performs the ratchet step. She generates a random ratchet key pair $rsk_{A,0}, rpk_{A,0}$ and considers Bob's signed prekey $prek_B$ as the initial ratchet key for Bob. Following our notation, this key is a "-1^{th}" ratchet key for Bob. From this data she can derive $rk_{0,0}, ck_{0,-1:0} = \mathsf{KDF}_r(\mathsf{DH}(rpk_{A,0}, prek_B), 2\lambda, MSK, cst)$ and $ck_{0,-1:1}, mk_{0,-1:0} = \mathsf{KDF}_m(ck_{0,-1:0})$.

As we are interested in the key exchange part, the description of X3DH (Figure 3.3) and the key schedule (Figure 3.4) are the main information we need. However, we give in Figure 3.5 a high level view of a protocol execution between Alice and Bob to help visualize how the asymmetric and symmetric ratchets alternate in the protocol.

3.2 The security of Signal

3.2.1 A segmented analysis

The security of Signal (TextSecure) was first explored in 2016 in [FMB+16]. In this work, the ratcheting process is not fully considered, the authors analyse separately the different components

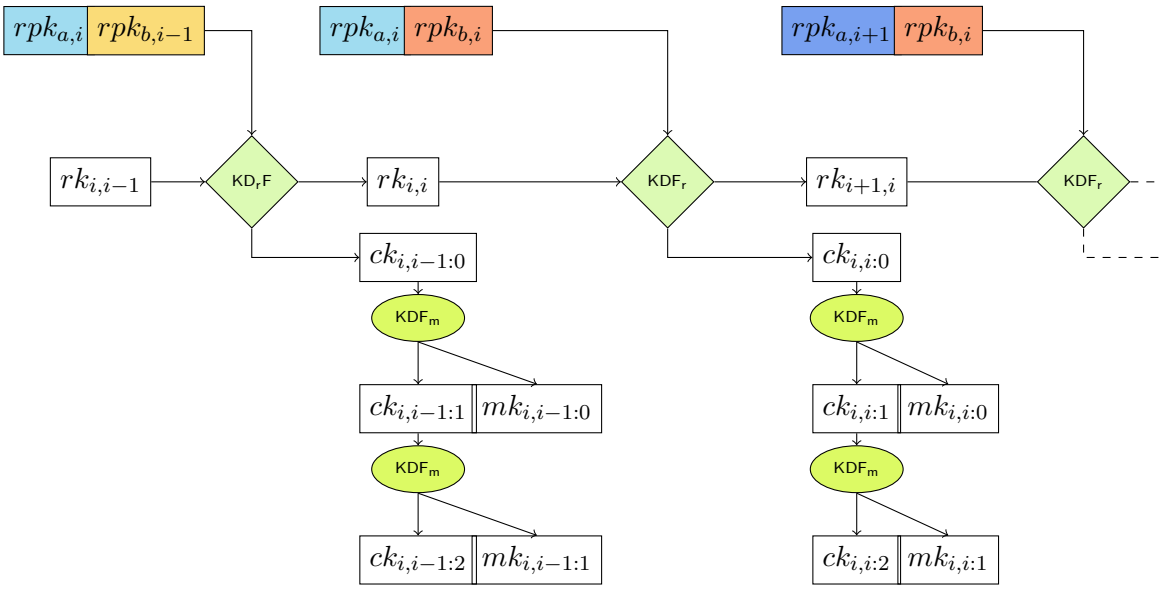


Figure 3.4 – The key schedule of Signal. The horizontal evolution with diamond KDF_r represents the asymmetric ratchet. The vertical evolutions with ellipsoidal KDF_m formalise the symmetric ratchets. (We only pention public ratchet keys in coloured boxes but the DH associates one of these public key with the secret key corresponding to the other.)

of the protocol. Firstly, they analyse the initial key exchange, identified as a cached one round key establishment. Back then (in 2016), X3DH was not specified and TextSecure implemented a variant of the protocol described in Figure 3.3. There was no mid-term key $prek_B$ but only one time ephemeral keys, and the derivation was always executed on the secret entry $epk_{B,j}^{sk_A}||pk_B^{esk_A}||epk_{B,j}^{esk_A}$ (written here on Alice's side). Their analysis takes place in a model that is situated half way between Bellare-Rogaway and the extended Canetti-Krawczyk. The adversary is given access to a RevealLongTerm oracle[6], but has no RevealState. He has no access to ephemeral secrets $eks_A, esk_{B,j}$ (apart from impersonating Alice or Bob and registering one-time keys on behalf of them). Their security result is based on the gDDH assumption, and they idealize the key derivation function as a random oracle. (As an *apparté*, we note that there has been limited works focusing on the X3DH protocol since this first analysis. A regain of interest seems to appear in the Post Quantum context, as reveal the recent [BFG+20] and [HKKP21].)

The authors clearly point out that "the multiplicity of different secret values established during the TextSecure protocol makes it difficult to correctly define queries revealing any long-, medium- or short lived values to the adversary", which in fact gives birth to a complex model in the complete Signal analysis provided in [CCD+17].

The authors then turn to the case of the encryption scheme (an encryption together with a mac) and to the key derivation function. The key derivation they consider corresponds to KDF_r followed by KDF_m (see Figure 3.6).

They prove the security of this construction given HMAC is a secure PRF and HKDF is modelled as a non programmable random oracle. In the analysis of Cohn Gordon, that we explore in more

[6] It is originally called Corrupt but it only gives access to the long term secrets, not to the state, hence corresponds to our definition of RevealLongTerm.

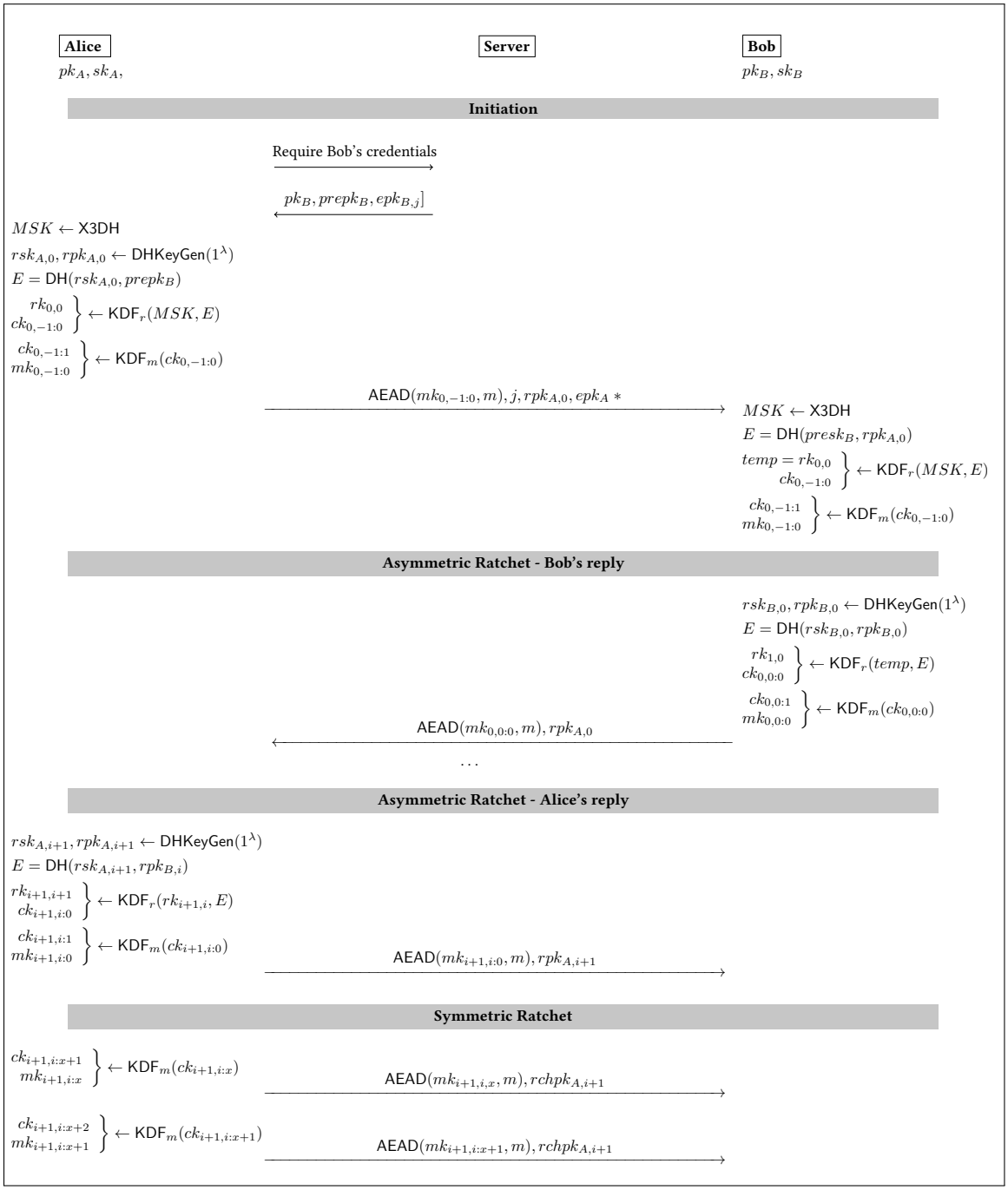


Figure 3.5 – Alice is the initiator of the conversation. AEAD is defined in subsection 2.3.4. X3DH, KDF_m and KDF_r are defined in subsection 3.1.2. An asymmetric ratchet occurs when Bob (then Alice) replies. Symmetric ratchets happen when Alice or Bob send several messages in a row. Note that the root key computed when receiving a fresh DH value from the peer (indexed with (i, i) on Alice's side, with $(i + 1, i)$ on Bob's side) is considered as a temporary value and the next root key immediately computed.

$\mathsf{KDF}_{TS}(\mathsf{DH}(rpk_{a,i}, rpk_{b,i-1}), rk_{i,i})$

1 : $rk_{i,i+1}, ck_{i,i:0} \leftarrow \mathsf{HKDF}(\mathsf{DH}(rpk_{a,i}, 2\lambda rpk_{b,i-1}), cst_0)$
2 : $mk_{i,i:0} \leftarrow \mathsf{HKDF}(\mathsf{HMAC}(ck_{i,i:0}, cst_1), cst_3, cst_4)$

Figure 3.6 – The TextSecure Key derivation function considered in [FMB+16].

details, both KDF_r and KDF_m are considered as random oracles[7].

3.2.2 The multi-stage model extended

The analysis by Cohn-Gordon *et al.* encompasses the whole Signal protocol - the initial key exchange and further ratchet - in a single multi-stage protocol. As we recalled in subsection 2.4.3, a stage is a chain of steps of the protocol that lead to a stage session key. In the case of Signal, the stage session keys are the message keys. They are the keys one wants to ensure the security of. They are the keys the adversary will be allowed to Challenge. Root keys and chain keys can be seen as intermediate computation values, but are not considered as session keys as they are not used to secure communications between Alice and Bob (inside or outside the protocol).

The main difference with Quic or TLS (the two other main protocols studied as multi-stage key exchanges), is that the sequence of possible stages is not linear. Once a stage is completed, *i.e.* once a message key is obtained, there are two options : either a subsequent message key is derived through a symmetric ratchet (if several messages are sent in a row by a same user). Or the subsequent message key is derived from an asymmetric ratchet (the first message each time a user replies). We represent in Figure 3.7 the two different kind of stages that lead to a message key. This motivated Cohn Gordon *et al.* to propose in [CCD+17] (and in the full version [CCD+16] that has significantly evolved since 2017) a multi-stage model with a tree of possible stages, where each branch corresponds to a different sequence of asymmetric/symmetric stages.

The adversary. Within this multi-stage framework, the adversary $\mathcal{A}$ controls the network and the message delivery as in the classical model. However, he is given fine grained corruption oracles. Firstly, he needs a RevealMidTerm in order to take into account the role of the signed prekeys that enable the asynchronous one round initial key establishment. Secondly, in addition to the RevealLongTerm and RevealSessionKey oracles, he can reveal the ephemeral diffie hellman values ($rsk_{A,i}$, $rsk_{B,j}$) through a RevealRandom oracle. This would correspond in spirit to the RevealEphemeral of [LLM07]. Thirdly, $\mathcal{A}$ can access the intermediate root keys and chain keys through a RevealState. Where in [LLM07], the author replaced the RevealState with a RevealEphemeralKey, [CCD+17] adopts both of them.

A Signal specific freshness definition. Considering the fined grained power given to the adversary, the freshness conditions are tailored for the protocol. As can be seen in Figure 3.7, the stages in Signal are not independent, they overlap each other. This results in a recursive definition of the freshness conditions. Let study the security of $mk_{i,i:0}$. It is easily seen that it is secure if $ck_{i,i:0}$ is. Now $ck_{i,i:0}$ is secure either if $rk_{i,i}$ is unknown to $\mathcal{A}$, or if both the ratchet secrets $rsk_{a,i}$ and $rsk_{b,i}$ are secure (the adversary is not allowed to reveal both a ratchet secret and the root key). But the

[7] The authors give an extended intuition of how it could be possible to without the ROM, relying on several PRF-ODH assumptions, as defined in [BFGJ17], some of which do not have clear reduction to classical assumptions.

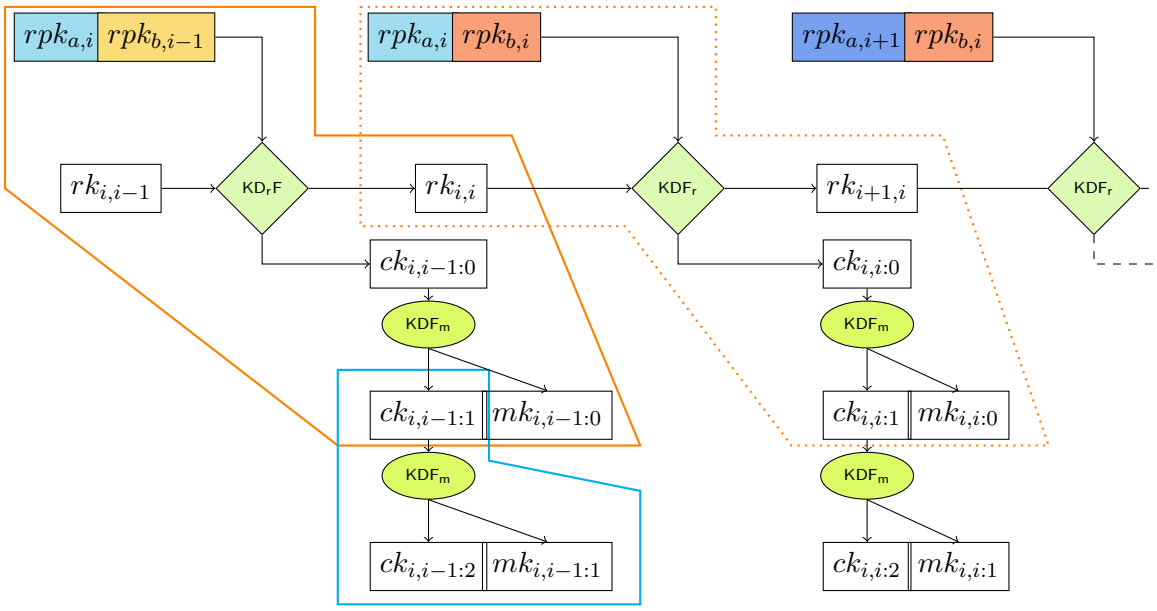


Figure 3.7 – The stages in Signal. All the steps that compose an asymmetric stage are surrounded in orange. The steps that compose a symmetric stage are surrounded in cyan.

adversary has two ways to corrupt $rk_{i,i}$. Either he directly corrupts the corresponding state (with a RevealState(i,i)) or he gets $rk_{i,i}$ from revealing the previous state and a previous ratchet secret.

The same way, the freshness of a message key in a symmetric stage depends on the freshness of the previous chain keys which also depends on the freshness of the corresponding asymmetric stage.

PFS. Firstly we consider PFS relatively to the long-term secret. If the identity key of one participant is corrupted, then Signal, even with the slight modifications considered in [CCD+17], reaches PFS. The long term key is only used during the initial X3DH, and this protocol is safe as long the adversary does not corrupt both a long term key and an ephemeral key (of either of the two participants). As ephemeral keys are signed, the unforgeability of the signature scheme shall prevent the adversary to insert malicious ephemeral keys, and PFS follows.

As recalled in section 2.4.1, in a constantly refreshed and long-lived session, PFS is to be understood relatively to the state. If the current state of a participant is compromised, then the adversary gets a root key (*wlog.* we suppose it of the form $rk_{i+1,i}$) and a chain key $ck_{i,i:,n}$. From the chain key $\mathcal{A}$ can not compute the previous message key of stage (i,i), due to the security of the key derivation function. From the root key (which is the next stage root key), the adversary can not compute previous root keys, hence neither previous chain keys and so past message keys are still safe. As we detail further on, $\mathcal{A}$ is forbidden by the definition of the freshness to insert its own ratchet values. This means that only the weak form of PFS is considered. Now consider the adversary could control the ratchet values. Given, at some time in the future, the stage at some stage and the long term keys, he will access past session keys only if he knows the ratchet secrets from the beginning of the execution. If there were an honest exchange between Alice and Bob, he would not access the following keys. This is where PFS and PCS meet ! It is important to note that, in the asynchronous setting, PFS can only be guaranteed with respect to the less advanced participant. If Alice answers

Bob with several messages, then these message keys will not be forward secure as long as Bob will not have updated his own state accordingly.

PCS. The asymmetric ratchet provides post compromised security, as it introduces fresh randomized values in the computation of session keys. The window of passivity required from the adversary is a way and back exchange between Alice and Bob. As for PFS, if $\mathcal{A}$ gets Alice or Bob's state, then he obtains a root key (*wlog.* we suppose it of the form $rk_{i+1,i}$) and a chain key $ck_{i,i}$. From then, if he compromises the future Diffie Hellman ratchet keys $\widetilde{rsk_A}$ or $\widetilde{rsk_B}$ then he can compute all the future keys. However, if at some time, Alice and Bob manage to exchange random ratchet values rpk_A^*, rpk_B^* without $\mathcal{A}$ knowing the corresponding secrets, then (considering *wlog.* that Alice starts):

- $rk_{j+1,j} = \mathsf{KDF}_r(\widetilde{rk_{j,j}}, \mathsf{DH}(rsk_a^*, \widetilde{rpk_B}))$ where $\widetilde{tilde}$ values are controlled by $\mathcal{A}$, is still known to $\mathcal{A}$.
- $rk_{j+1,j+1} = \mathsf{KDF}_r(\widetilde{rk_{j+1,j}}, \mathsf{DH}(rsk_b^*, rpk_a^*))$ is unknown to $\mathcal{A}$. The protocol has "healed itself".

In the above scenario, one does not consider the case when the adversary introduces its own Diffie-Hellman randoms. As explained in the next paragraph, this kind of impersonation is directly ruled out by the model. Another limit to this formalisation of the PCS is that it requires the state (the root key and chain key) to be separated from the ratchet random values. As they are all temporary values, which need to be kept for a nearly equal amount of time, on can imagine that an implementation would treat them accordingly. Now, if the adversary can access both the root key and the ratchet random in a single oracle query, he will be able to compute the following root key. However, from then the previous reasoning remains valid. If he does not compromise the new state (the new ratchet randoms) then the next session keys are safe. Recently, Cremers *et al.* studied in [CFKN20] the PCS provided by real world implementations of Signal. They highlight that the tolerance of desynchronization necessarily induces a loss of PCS (this can be seen as a parallel to the fact that tolerating out-of-order message weakens PFS), by enabling cloning attacks.

The final security statement on the security of Signal is given in Theorem 3.1. It is interesting (but not surprising) to see that the security of the whole protocol rely on the same assumption that the sole initial key exchange as studied in [FMB+16].

Theorem 3.1. *The Signal protocol is a secure multi-stage key exchange protocol under the* gDDH *assumption and assuming all KDFs are random oracles.*

The security reduction is particularly non-tight (and recognized as such by the author) as their proof induce a loss factor of $n_s n_p^2$ on the $gddh$ term. Considering an application as Signal that has billions of users and more sessions, this boils down the security. On the contrary, instantiating the application with parameters that achieve a 'certain) amount of security relatively to this statement would result in an impractical protocol. This paradox is studied in [CCG+19] (and further in [JKRS20]), in which the authors underline that the traditional game hope strategy for key exchange which consists in guessing which session will be targeted is inherently non tight. They propose new designs for key exchange to overcome this problem but their strategy is not applicable to Signal.

Reality *vs.* idealisation The analysis of Cohn-Gordon *et al.* tries to be as close as possible as the real implementation. However, the author conceded some modifications that we detail below.

Authenticating the ratchet values. In the model, message keys are not used within the protocol, hence the "stop-and-go" mechanism introduced in [FG14] is not necessary. However in the real implementation, the authors note that the message keys also authenticate the ratchet material sent to each other *i.e.* that they are used within the protocol itself. They modify the protocol and consider the ratchet material is sent unauthenticated. In order to obtain a security statement however, they enforce authentication in the freshness conditions of their security model, by declaring fresh an update randomness only if it has been produced and sent by an honest peer (preventing *by fiat* the adversary to insert its own ratchet randomness). With an authentication mechanism, then the PFS and PCS properties are achieved more clearly (*i.e.* without requiring the adversary to be passive, the security of the authentication should prevent impersonation. This is not trivial. For instance, the authentication mechanism should evolve to provide PCS with regard to long term keys. However it shall be independent from the root key otherwise no PCS is obtained at all. If the adversary compromises the state, then he obtains both the root key and the solution to introduce its own refreshing values.)

Out-of-order messages. In a real world implementation, one cannot be sure that the network will convey the messages in the right order. Bob can receive the third message from Alice before the first and so on. To deal with this, Signal's implementation (WhatsApp implementation is not available but one can suppose this is a common choice) keeps memory the message keys corresponding to skipped messages. This seriously harms the PFS property. In their analysis, Cohn Gordon *et al.* suppose that the messages arrive in order. We will see in section 3.3 that only one security model for the ratchet takes into account the out-of-order messages.

3.2.3 On other security properties of Signal

Key indistinguishability, together with PFS and PCS are the main classical properties one expects from a key exchange. But several works have been interested in other aspects. We look at two of them, that we consider of non negligible interest.

Deniability. At a first glance, the definition of deniability seems to contradict the goal of the authentication. Roughly, the idea is that Bob shall not be able to convince an external party - the judge - that he conversed with Alice. Even if he is very bad and misbehave during his exchange with Alice in order to trap her. If Bob and Alice exchange messages protected by a symmetric key mechanism, then deniability is offered, as Bob can have generated the whole conversation itself. The most difficult part is to ensure that the - nearly - unavoidable AKE needed to share the symmetric key is itself deniable. Deniability has long been assumed for Signal X3DH protocol. The Systematization of Knowledge of Unger *et al.* attributed three deniability properties to TextSecure: message unlinkability, message repudiation and participation repudiation. It has only recently been proven, by Vatandas *et al.* in [VGIK20]. To be precise, the authors show the offline deniability of Signal, when the judge exchanges with Bob after the key exchange session is over. Online deniability, when the judge can influence Bob's behaviour during the session is studied in [UG15]. Vatandas *et al.* follow the definition of deniability given in [DGK06], based on the simulation paradigm: it requires the existence of a simulator that, in interaction with Bob, generates transcripts that are computationally indistinguishable from real transcripts between Alice and Bob. Intuitively, implicitly authenticated key exchange protocols (in which the authentication is obtained because

the long term key is implied in the derivation of the session key, but where no formal authentication mechanism such as a signature is at stake in the transcript) appear as good candidates for deniability. And X3DH is one of them. The author show that the intuition is good, even if, under some extreme conditions (if the group chosen for the key exchange happens to be a group where the DDH is easy), non deniability can be proven. However, such an hypothesis also defeats the security of the key exchange (their case study is MQV []) and will be more rarely encountered. The interesting part of their work is that they include the final session key in the simulation result, such that they can prove the deniability of the whole Signal protocol, on top of the deniable X3DH.

About the trusted server. The X3DH initial key exchange relies on the Signal server to distribute the correct key, to initiate the communication. In theory, one shall identify - physically with QR code, or with a phone call for instance - the long-term key of each of its correspondent, any time it changes. In their Systematization of Knowledge ([UDB+15]), Unger *et al.* identify key fingerprint verification as introducing "severe usability and adoption limitations: users have to perform manual verification before communicating with a new partner to ensure strong authentication." This step is optional, hence augmenting usability, but this means that the confidence in the server is high. In [CDGM19], Chase *et al.* propose a verifiable key directory (VKD) construction, "which allows users to monitor the keys distributed on their behalf". Their work formalises the notion of VKD, that was concretely implemented by [MBB+15] or [TD17] for instance and proposes a construction based on zero-knowledge proofs. At a high level, the idea is that the server keeps a directory of the key updates together with proofs that the update was performed by the appropriate user. When requiring the key of Bob, Alice will receive the key together with the history of update proofs such that she can be sure that the key is in fact Bob's one.

Another direction was proposed in [BBB+19], replacing the trusted server with identity based (ID-based) cryptography. The idea of instantiating the public key with any arbitrary string was introduced by Shamir in [Sha84] and a first practical encryption scheme was given in [BF01]). The SAID protocol proposes to replace the central Signal server, that has to be online for any session initialization, with an ID based key distribution center that only has to be online for the registration of a user [8].

Finally, the recent work of Ruggeri *et al.* [RCF+20] constructs a block-chain based version of the X3DH, eliminating the central server.

3.3 From a protocol analysis to a formal cryptographic primitive

The ratcheting technic could have remained a special case of multi stage. But the work of Cohn-Gordon *et al.* throws light on the need for some more abstraction. As noticed by the authors, when they started the analysis, the security properties that were claimed for Signal were not well defined. This blurred definition of security somehow emphasized the need for a new primitive: which security properties are we entitled to expect from a protocol like Signal ? To what extend a process as the double ratchet shall improve the basic security of a key exchange ? Once again, another interesting question between those lines is: does the game is worth the candle ? More formally, identifying the security benefits of a primitive in an appropriate model also helps deciding which degree of complexity in a protocol (on the implementation and performance's side for instance) is acceptable to reach them.

[8] This work also proposes a model to analyse their alternative Signal protocol in the ACCE setting. Their model is highly related to the ID-based setting and we do not include it in the state of the art that we provide in the next section.

3.3.1 The different propositions

Several works have set out, sometimes concurrently, to formalise the ratchet and its security. In the following, we compare the propositions of the following: [BSJ+17], [PR18b], [JS18b], [DV19], [ACD19], [JMM19] and [CDV21]. We first turn ourselves to the case of [PR18b], [JS18b], [DV19], and [JMM19], as their goal is to maximise the security one can expect from a ratcheted key exchange. In a second time, we examine [ACD19] and [CDV21], that are closer to the actual Signal.

Bellare, Singh, Jaeger, Nyayapati and Stepanovs put forward, in [BSJ+17], the first abstract formalization of the ratchet as a cryptographic primitive. They focus on a one sided ratchet primitive: the two participants have well defined roles, there is one sender and one receiver, and only the sender is actively involved in the ratcheting process. As such, the state of the receiver's state remains as sensitive as in a traditional long-term key based key exchange. Despite the interest of this seminal work, we will focus in the following comparison on the schemes that propose a bi-directional version of their ratchet solution.

Following this work, Poettering and Rösler propose in [PR18b] (full version [PR18a][9]) their own version of a single directional ratchet key, then extend it to a sesquidirectional form (where the receiver can also inject some new entropy but his Send messages do not lead to a new session key. The role of the sender and the receiver are kept asymmetrical). They finally propose a bidirectional ratchet key exchange, that can easily be seen as the combination of two sesquidirectional versions: one where the sender is A, the other where the sender is B. They are attached to capture the highest level of security that one can expect from such a primitive. Hence their strategy is to give full powers to the adversary, then to exclude only the unavoidable attacks in the freshness conditions, as we will detail in section 4.3.2.

The proposition of Jaeger and Stepanovs ([JS18b], full version [JS18a]) follows the same path, and we will see that their model encompasses the same security level as [PR18b], identified as fine-grained compromise. Their work is however slightly different as they do not focus on the key exchange feature but on the more global channel point of view (*cf.* section 2.4.3). As such, their primitive is described with encryption and decryption algorithms (instead of sending and receiving ones as detailed below). Yet, the comparison of the powers offered to the adversary is still valuable.

At Eurocrypt 2019, Jost, Maurer and Mularczyck also formalises in [JMM19] (full version [JMM18]) a bidirectional messaging solution that achieves almost-optimal security, situated just below [JS18b] and [PR18b]. Concretely, this means they have to exclude one more specific attack in addition to the unavoidable ones. Thanks to this little relaxation, they obtain a somewhat more efficient scheme.

Concurrently, Durak and Vaudenay also proposed in [DV19] (full version [DV18]) a formalisation of a bidirectional asynchronous ratcheted key agreement (BARK). As for Poettering and Rösler, they first propose an uni-directional version of their protocol, where one participant is a sender and the other a receiver only) before doubling the uni-version to get a bidirectional protocol. They achieve a sub-optimal security, that is below the above almost optimal one. They introduce an interesting property, that they call Recovery, that stipulates that a protocol should not recover from an impersonation.

In the following, we adopt a single syntax that can fit with any of the above proposition. Let it be for the primitive formalisation or for the oracle that the adversary can access. The description of the adversary's power or of the freshness conditions will then reveal the differences between the considered models.

[9] We always specify the full versions for the following work as they contain significantly more information than the conference ones.

3.3.2 Formalizing a RKE model

The primitive. About half of the propositions formalised ratcheting in a key exchange protocol, while the other half chose a messaging protocol. We focus on the key exchange formalization of the primitive. Firstly, a key exchange primitive appears to us as more general, as a secure channel can be composed (carefully) of a key exchange and an encryption scheme. Secondly, in the previous works detailed in section 3.2, Signal and its predecessor TextSecure and OTR are analysed as key exchange, hence, focusing on the key exchange part facilitates the comparisons.

A ratchet key exchange (RKE) is a protocol between two participants, A and B, composed of three algorithms:

- Init, that, on input the security parameter returns a state for each participant s_A and s_B.
- Send, that, on input a state s_U, returns an updated state s'_U, a session key k and an update information upd (in [PR18b] this update information is directly mentioned to be a ciphertext, which is more adapted to their key encapsulation based construction. We keep the update version of [BSJ+17] as we consider it more general.)
- Receive, that, on input a state st_U and an update information upd, returns an updated state st'_U and a session key k or an error symbol $\perp$.

One thing to note is that the above description quits the traditional "message driven" description of a key exchange protocol and assigns specific roles to the sending and receiving action (this is a reason why the channel versions are similar to the key exchange ones, except that the Send algorithm outputs a ciphertext and the Receive algorithm outputs a plaintext message).

The environment. Another major point in the above description of a RKE is that it considers an initialisation step that provides both participants with their initial state. If the participants were to share a common secret, it would be provided by the Init algorithm. That is, the ratcheted key exchange formalised as above does not consider the initial authenticated key establishment between Alice and Bob. This means that the environment for the security model does not have to consider several participants and concurrent sessions, but only focuses on a single protocol execution between "fixed" Alice and Bob.

The adversary. The adversary has access to the oracles to perform the protocol: OSend, OReceive. He can reveal the current session key on U's side with a RevealSessionKey(U). He can access the whole state of a participant through a Expose(U). It could be assimilated to the Corrupt oracle by [CK01b]. However the term Expose(U) may be less associated to a long-term key value than Corrupt is. As the initiation procedure is abstracted in the RKE primitive, there may be no individual long term key at stack in the protocol. The models of [JS18b] and [JMM19] allow the adversary to access the randomness (a full control in the first case, a reveal just before a message is sent in the second), which can be seen as an equivalent of the RevealRandom of [CCD+17]. However, in Signal, the random coins were directly included in the key derivation whether in [JS18b] and [JMM19], the main consequence of a randomness leakage is the loss of confidentiality of publicly encrypted messages.

Finally, the adversary can require to be challenged, on the sender or the receiver side, with a Challenge(U) query. For the key exchange version, he will receive either the real key, or a random one. The channel versions of [JMM19] and the Secure Messaging of [ACD19] both adopted the

left-or-right indistinguishability. In [BSJ+17], [PR18b], [JS18b], [JMM19] the adversary may require multiple challenges. One consequence is that the freshness of the query shall be verified in-line while in a single challenge version, one can delay the verification to the end of the experiment.

Epochs. Epochs were introduced in [PR18b] and help the Challenger to keep trace of the updates that happened during the protocol. Within an epoch, several keys may be computed by the same user. A new epoch starts when a new entropy is introduced and ends when this entropy is acknowledged by the partner. Keeping an eye on Signal, an epoch can start with an asymmetric ratchet while several symmetric key derivation can happen within an epoch. The epoch counter plays an important role in the freshness. In [PR18b] (and so would it be in [JS18b] or [DV19] if they adopted the epoch vocabulary), the Challenger maintains one epoch counter per participant, that is incremented with every sending operation, independently from the receiving ones. This can only be realised if Alice and Bob each maintain and update their own personal key stream, independently from each other. In [ACD19], that we detail later on, and so would it be in Signal, Alice's epoch counter is incremented by a sending only if it comes right after a receiving. This traduces a ping-pong pattern in which a single key stream is updated by both participants.

Matching. As there is no multiple sessions, there is no notion of matching, as defined in subsection 2.4.2. However, there is still the need to know whether or not both the participants have computed the same key (to know whether one can be corrupted and the other challenged, essentially). Hence the model keeps trace of the synchronisation of the two participants. If the protocol executes normally, then Alice and Bob are always synchronized. If the adversary injects its own message in the receive oracle (in place of a legit message produced by the send), then Alice and Bob are desynchronized. If this forgery is not issued directly after a state exposure, (which is a trivial attack forbidden by the freshness requirements) then the adversary wins. Synchronization can be formalized with a `sync` flag in the code based description of the game as in [PR18b] (`hijacked` in [JMM19]). Durak and Vaudenay chose a formal definition of matching. The victory of the adversary in case of a forge is considered separately, in a specific FORGE game. If more explicit, the first solution only spots out-of synchronisation due to the receiving of a forge. The matching definition compares that all the messages received by Alice where indeed produced by Bob. But it also checks whether there had been no forgery on Bob's side at the time he sent his message. That is, a previous forgery on Bob's side would also make Alice out-of-sync (and not only Bob). However, both definitions merge as in [PR18b], any forgery received by Bob automatically defines his further message as forgeries, driving Alice out-of-sync immediately after. In [JS18b], synchronisation is not properly formalised but equally taken into account by recording the forgery events.

Correctness. Two different forms of correctness are proposed in those work. Both stipulate that two synchronized participants should compute the same keys (or, in the channel models, the receiver should be able to decrypt the ciphertext sent by its peer). Their difference is on the possibility or not to resist to a misformed (malicious or not) update. We adopt the notation of [JS18b], $\mathrm{Corr}\perp$ and Corr, to distinguish both notions but we give the definition for a key exchange. In the first definition ($\mathrm{Corr}\perp$), upon receiving an incorrect update (*i.e.* of the Receive algorithm returns $\perp$), the state of the receiver is somehow erased and the correctness game is over. This is the one adopted in [PR18b]. In the second version (Corr), the receiver maintains its previous state when receiving an incorrect update and is still capable of treating an incoming correct update. This is the one adopted in [BSJ+17], [DV19], or [JS18b].

The latter explains that it is trivial to switch from one definition to the other (in fact, in a channel construction for instance, one can decide whether or not to close the channel upon a bad decryption). However, the two notions capture well distinct properties. The Corr implies a form of robustness (and is denominated robustness in [BSJ+17]) of the scheme, that can carry on regardless a wrong event. On the contrary it can be seen as a weakness as the scheme will not react to an attempt of corruption for instance. Consider for instance the concrete scenario given in [JS18b]: the adversary $\mathcal{A}$ corrupts then impersonates Bob. Alice will receive messages from $\mathcal{A}$, thinking they come from Bob. Imagine now the honest Bob succeeds in sending a message to Alice. Alice will not be able to process this update. In a $\mathrm{Corr}\perp$ secure protocol, this directly cuts off the communication, while with the Corr, it does not reveal that something went wrong. On another side, implementing the $\mathrm{Corr}\perp$ correctness can seriously damage the efficiency of the scheme as every such misformed update requires to reopen a channel (including the costly initial non interactive key agreement). This opens the way to denial of service attack. This is the same dilemma again, security confronting real life efficiency.

3.3.3 According on PFS...

All the above models exclude the trivial attacks. Only [JS18b] and [PR18b] put no other restrictions. The differences - that may appear small - in the expression of the freshness conditions traduce differences in the expected PFS and PCS levels.

Trivial challenges. The adversary is forbidden to reveal a session key and then challenge it. Conversely, the adversary is forbidden to challenge a key and immediately reveal it. If the participants are synchronized, this condition extends to the peer naturally. In the channel model, the adversary has no session key reveal algorithm. However those trivial challenge conditions also appear in the state reveal restrictions.

PFS. A state exposure does not compromise past computed keys. The state exposure of Alice will compromise the confidentiality of all the updates that have been sent by Bob and not already received by Alice. The keys obtained through those updates are set unfresh (the other way round, in the channel set up, the exposure of Alice is forbidden while there are challenge messages from Bob still in transition). If Bob is synchronized, then the corresponding sending keys are set as unfresh. One can not do better for, if Alice is supposed to be able to process those updates (to derive the corresponding key or to decrypt the messages), then it must be that her state contains all the needed information to do so. Hence, PFS necessarily relates to the oldest accounted update. It is then intuitive that, when out-of-order messages are authorised, a lost update can postpone PFS.

3.3.4 ...but differing on PCS

One can detail two facets of PCS, that are deeply related. Suppose Alice's state has just been compromised. Firstly, one can legitimately wonder about the confidentiality of the future messages/keys. Secondly, it is necessary to study the question of the authentication: when can we be sure that the adversary can not trivial insert its own update information (on Bob or Alice's side). As said before, these two questions can not go without another, but may not be impacted equally. In this paragraph, we study the different flavours of PCS provided by the consecutive (and sometimes concurrent) works described in subsection 3.3.1. We chose to focus on the description of two constructions, [PR18b] and [DV19], to underline how a small step in the PCS requirement can induce a gap in the efficiency.

Fine grained PCS. The most demanding models are the ones of Poettering and Rösler, and Jaeger and Stepanovs, that we identify as fine-grained PCS. In these models, the Exposure of Alice's state:

1. identifies as unsafe all the future updates (message) sent by Bob, up to the next message sent by Alice itself. As for PFS, if Alice remains passive (only receives information) then her state only evolves with the receiving messages. As the initial state exposure enables the adversary to process all those updates, then one can not avoid those keys (messages) to be known to the adversary;

2. makes all the future messages sent by Alice trivially forgeable. There is nothing to prevent from this attack as Alice's state necessarily contains all the elements to authenticate herself.

3. shall not harm the confidentiality of future updates sent by Alice (the receiving keys of Bob). In Jaeger and Stepanovs, this confidentiality is lost only if the adversary also compromises the random elements on Alice side.

4. shall not break the authentication of Bob's message.

If $\mathcal{A}$ remains passive (does not use [2]), then by [4] Alice shall be ensured that the future messages received from Bob are honest (even if $\mathcal{A}$ can read them ([1]). And, thanks to [3], as soon as Alice sends an update (and that Bob acknowledges it), then the security is regained: this is the healing feature. The main difference with Signal is that healing is obtained instantaneously, while it required Bob to reply with its own update in Signal.

Now, if $\mathcal{A}$ is active and impersonates Alice (using consequence [2]). Then Bob is considered as out of synchronisation. In particular, even an exposure of Bob after the impersonation shall not damage the security of the potentially future keys/messages generated by the honest Alice. This specific post-impersonation security requires for both participants to be able to update already transmitted keys accordingly. We detail this process below when comparing to the weaker sub-optimal PCS requirement. Both constructions proposed by [PR18b] and [JS18b] are based on hierarchical identity based encryption (HIBE). This evolution of IBE (already mentioned in subsection 3.2.3)) introduced in [GS02] is naturally related to updatable encryption as it makes it possible to create successive secret decryption keys for related strings. But this mechanism is costly. For instance, in [GS02], the size of the ciphertext and the complexity of the decryption grow linearly with the number of updates. In a messaging protocol with long living sessions that require regular refreshing, this can not be considered as a practical solution.

Sub-optimal PCS. The construction proposed by Durak and Vaudenay is somewhat less optimal than the previous fine grained PCS. The main difference is that an exposure on Alice side breaks the authentication both on Alice and Bob's side.

Where do this difference comes from ? We elaborate on the construction of [PR18b] and [DV19] as they are close, to understand where the difference in the PCS lies. In both case, Alice and Bob both maintain two keystreams: one for the messages where Alice is the sender and Bob the Receiver, and a second one for the reverse communications. These channels are based on asymmetric encryption and a signature scheme in [DV19], while they are built from a key updatable KEM, a one time signature and a MAC in [PR18b]. Each time Alice sends a message, her state is composed of several sending states, (corresponding to the epoch not acknowledged by Bob yet) and several receiving states (corresponding to updates received by Bob and not acknowledge yet).

The sub-optimal version. In [DV19], receiving states are composed of a public signature key and a secret PKE decryption key while sending states logically contains a public PKE key and a secret

signature key. On sending a message, Alice generates a new session key k, a brand new receiving state $\mathtt{st}^A_{rec}$ for her (this corresponds to the generate algorithm in Figure 3.8), that she appends to her receiving states directory and the corresponding sending state $\mathtt{st}^B_{snd}$ for Bob. In parallel, she generates a new sending state $\mathtt{st}^A_{snd}$ and a corresponding receiving state $\mathtt{st}^B_{rcv}$ for Bob. Then $\mathtt{st}^B_{snd}$, $\mathtt{st}^B_{rcv}$ and the new session key k are signed and encrypted successively under each sending state, forming an "onion structure"[10]. This structure serves as an acknowledgement of all the previously received messages. Finally $\mathtt{st}^A_{snd}$ replaces the most recent sending state in Alice's sending directory (it is not a new state but an update, this is the renew step in Figure 3.8). Always keeping an eye on Signal, this would correspond to the symmetric ratchet, while not symmetric. If Alice sends several messages in a row, she does not generate a new sending epoch with each message but only update keys within the corresponding epoch. When Bob receives a messages from Alice, he peels the onion and gets his corresponding updated receiving state $\mathtt{st}^B_{rcv}$, his brand new sending state $\mathtt{st}^B_{send}$ and the new session key k. It goes the same way round when Bob sends a message and Alice receives.

The fine grained version. In [PR18b], the sender also needs a public encapsulation key and a private signature key, and the receiver the corresponding private and public keys. Here we detail some structural differences that do not have an impact on the PCS level. Firstly the signature keys are not accumulated, only the most recent one is kept in memory. With each message, Alice generates a new signing pair, keeps the private one and transmits the corresponding public key to Bob. The acknowledgement of previous received updates is constructed only on the encapsulation scheme. Before sending a message, Alice will encapsulate to every public key she has in her sending state directory. From the keys output by all the encapsulation, she derives a chaining value (that provides passive authentication as the root key in Signal), the session key and a new (public, private) encapsulation key pair. The private key is erased while the public key replaces the most recent public encapsulation key in memory : this is the renew step in Figure 3.8. This way, only public keys needs to be transmitted to Bob (but the derivation implies that the proof is settled in the random oracle model). When Bob receives a message, he gets the public future sending information $\mathtt{st}^B_{snd}$ and he can derive the same shared secret so that he computes the private encapsulation key $\mathtt{st}^B_{rcv}$ corresponding to $\mathtt{st}^A_{snd}$. Again, roles are interchangeable. As we said, the structure differs but the result is very similar: a new receiving state st^A_{rcv} (and the corresponding st^B_{snd}) is generated while the most recent sending state st^A_{snd} (and the corresponding st^B_{rcv}) is renewed.

And now, we come to the main difference. In the receiving process, Alice compares the epoch E considered in the message she received (the most recent decapsulation key she needs) and the maximal epoch, say $E + k$ she has in memory. The difference k corresponds to the number of messages that she has sent but that Bob has not processed yet. With each of this message, Alice had generated a private decapsulation key $\mathtt{st}^A_{rcv}$ and sent the corresponding public key to Bob: this is the generate step in Figure 3.8. Now, as Bob has not taken them into account yet, Alice secures these keys by updating them, thanks to the updatable feature of the KEM. Every message received from Bob that does not take epoch $E + \ell, \ell \leq k$ into account leads to the update of the corresponding private decapsulation key for Bob. When Bob finally processes the message of epoch $E + \ell$, he has to count how many messages he has sent in the period of time [Alice sent epoch $E + \ell$ - now] and updates the public key of epoch $E + \ell$, that he has just received, as many times as necessary to be

[10] Actually the onion encryption is optimized and instead of encrypting successively in a for loop, the previous ciphertext becoming the next plaintext, they generate as many symmetric keys k_i as there is onion layers (say ℓ), encrypt one key in each layer and finally encrypt the plaintext symmetrically under $k = k_A \oplus \cdots \oplus k_\ell$. This saves bandwidth as it uses asymmetric encryption only for key encapsulation. However, this does not interfere with the PCS property and we do not elaborate more on this.

synchronized with Alice. We give a simplified view of this process in Figure 3.8. This asymmetric updating process *a posteriori* (public and private keys are updated separately, after they have been created) is the one that requires updatable asymmetric primitives, with public update information (as Alice and Bob must be able to perform the update without receiving any information from each other). In [PR18b] it is a key-updatable KEM and in [JS18b] key-updatable signature and encryption schemes.

Now, consider the attack in green in Figure 3.8. If Bob is exposed, then impersonated, the update synchronization is defeated and Alice state is useless. Without the update mechanism, exposing Alice after an impersonation potentially compromises all the unacknowledged states.

Almost optimal. The PCS modelled by Jost, Maurer and Mularczyck in [JMM19] is situated just below the fine grained PCS. We detailed it in third position only because it is simpler to appreciate the differences with the above description in mind. Their model also offer post-impersonation security: if Alice is exposed and Bob receives a trivial forgery, then:

1. Bob's authentication shall not be defeated;
2. the confidentiality of future messages sent by the honest Alice shall remain even under the exposure of Bob's state.

But those properties are only achieved if the last message sent by Alice before the impersonation remains confidential. For [1], this is because, as in [DV19], Alice sends the next secret signature key to Bob. To obtain [2], [JMM19] propose the following: whenever she sends a messages, Alice encrypts it with sending state corresponding to the last update received from Bob. But she also over encrypts it with an ephemeral PKE public key pk^{eph}. She further generates a new ephemeral encryption key from a secret seed z, and sends z to Bob. When Bob receives a message from Alice, he derives the corresponding $sk^{eph} = \mathsf{PKE.Gen}(tr||z)$ (where tr is a hash chain of the incoming transcript), he knows that all further incoming messages will have to be over decrypted with the newly generated sk^{eph}. If $\mathcal{A}$ impersonates Alice, then when Bob receive $\mathcal{A}$'s message, he will erase his previous ephemeral decryption key. Further messages from Alice will be protected as Bob's state does not contain the correct over -decryption key anymore. As the seed is secretly sent with each message, the security holds only if the last message sent by Alice was confidential[11]. The main advantage is that their secretly updatable scheme only relies on classic public key cryptography.

Finally, we gather in Figure 3.9 the different properties of the four schemes detailed above.

Whether or not they require HIBE, the above schemes are particularly non efficient if the communication does not regularly alternate between Alice and Bob. For HIBE, this deepens the hierarchy tree while in [JMM19], the number of asymmetric update operation is linear in the number of messages received in a row. Finally, in the onion structure of [DV19], this augments linearly the number of asymmetric encryptions needed for each message. Remark that, even without the post-impersonation security, the loss of efficiency is unavoidable, as obtaining an immediate healing (without waiting to receive a response) imposes to replace the symmetric ratchet step in Signal by a public key ratchet.

[11] In order to minimize the consequences of a randomness exposure, they propose a scheme where the updated encryption key is mixed with the previous one. As such, a randomness exposure breaks the confidentiality only if combined with a previous state exposure. However, if the adversary is in position to expose, then it is not improbable that he is able to perform two state exposures in a row, the first with a randomness leakage of the next message sent by Alice, the second just after, followed by an impersonation.

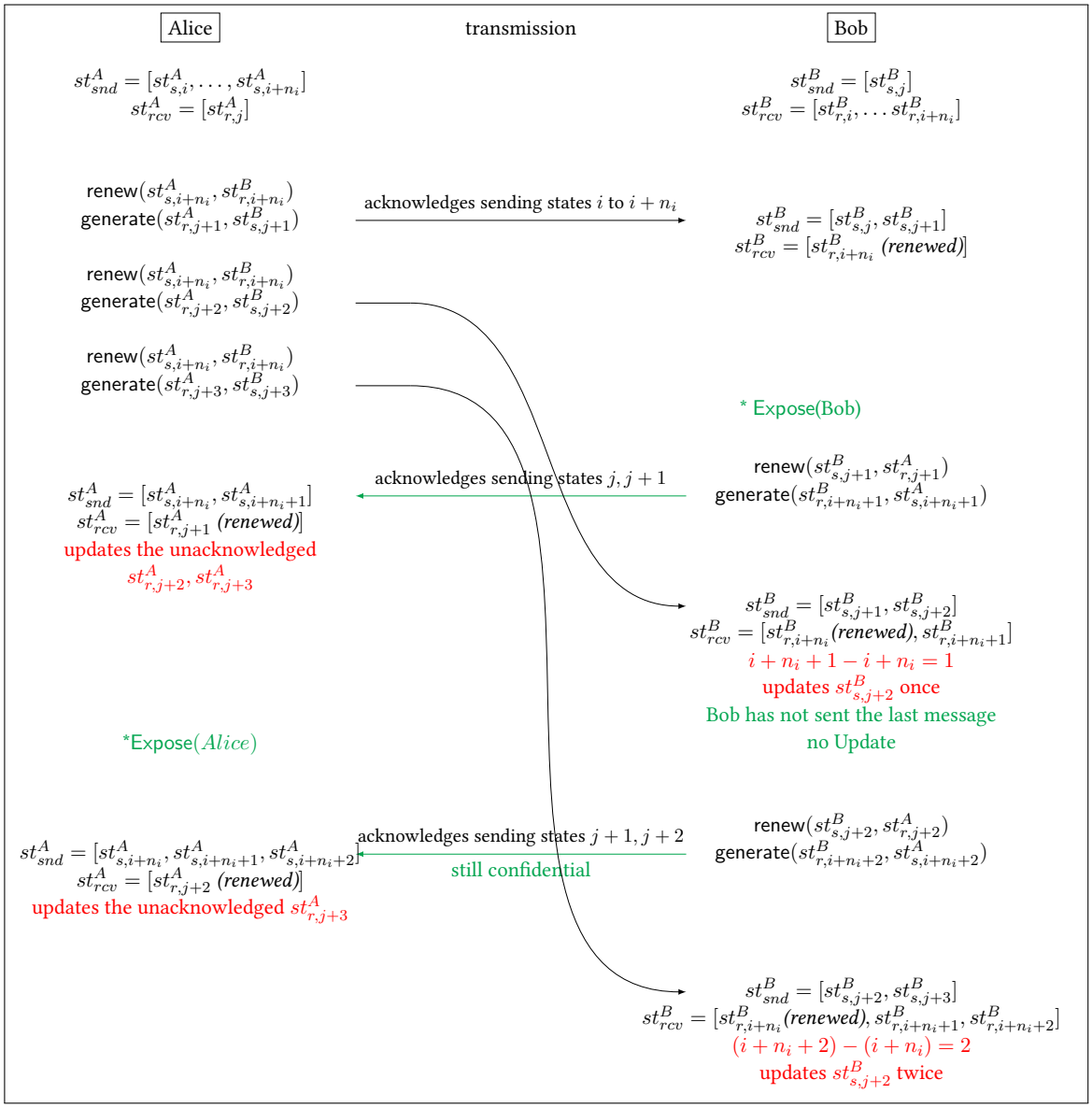


Figure 3.8 – A simple view of the updating process. The renew and generate algorithm corresponds to steps defined in section 3.3.4. In red, we mention the updating of the already created but not acknowledged yet keys. This process requires updatable public key primitives, as an updatable KEM in [PR18b]. The elements in green correspond to the attack that succeeds only without the update

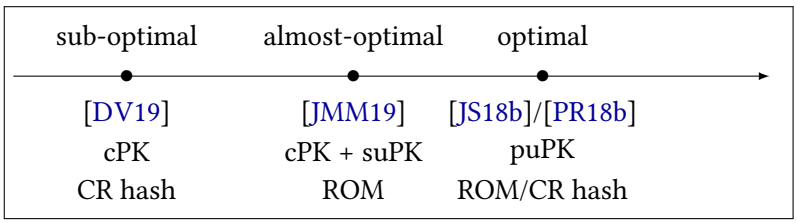


Figure 3.9 – Positioning of the different proposition in terms of security. cPK stands for classical public key cryptography, suPK for secretly updatable public key cryptography and puPK for publicly updatable public key cryptography. We also mention whether the security proofs are given in the plain model (settled on collision resistant (CR) hash functions) or in the ROM.

3.3.5 Signal like

Two propositions, among them a very recent one, are dedicated to protocols close to the actual Signal. In [ACD19] (full version [ACD18]), Alwen, Coretti and Dodis propose their own formalisation of Secure Messaging. They show that one can realise a secure messaging from two sub primitives: a continuous key agreement (CKA) on the one side and a forward secure authenticated encryption with additional data (FS-AEAD) on the other side. For this latter primitive, the authors propose a generic construction from a Pseudo Random Generator (PRG) and an AEAD scheme, that matches the Signal implementation when PRG $=$ HMAC based on SHA_{256}, while the former is implemented with a KEM. Their proposal is close to the actual Signal protocol and they recognize that they do not seek for the most fine grained post-compromise security. They formalise a new property called immediate decryption (*cf.* subsection 3.3.6). In their proposal, the CKA provides (the equivalent of) the asymmetric ratchet while the forward secure encryption abstracts the symmetric one. At first glance, the CKA could be seen as the proper equivalent of the other previous propositions (that are mostly asymmetric ratchet). However, the CKA model is very restrictive (it does not consider authentication, as the adversary is supposed to be passive and imposes a ping-pong order for the Sending oracle calls) and designed to be combined with a symmetric primitive to achieve a higher security level. Considering the whole secure messaging model, PCS is not as sharp as the previous work, for an exposure of one participant, say Alice, has consequences on both sides, on the confidentiality and the authentication side. The author introduce a PCS coefficient, Δ_{SM} that relates to the number of "way-and-back" messages necessary to heal from an state exposure. For the confidentiality, the number of messages required to recover depends on when the exposure takes place (was the exposed participant sending or receiving). But for both the sender and the receiver, it takes at least Δ_{SM} way and back messages (and at most $\Delta_{SM} + 1$) to recover. Authenticity only requires Δ_{SM} way and back messages, in any case. In the actual Signal protocol, $\Delta_{SM} = 3$, meaning the third message[12] after a compromise is always safe. This shared minimal cost reveals a shared state between the sender and the receiver. In their proposed construction, they show that replacing the "half-ratchet" of Signal (that updates only one half of the Diffie Hellman computation with each asymmetric ratchet) by a proper KEM reduces this coefficient to $\Delta_{SM} = 2$ (moreover, considering a post quantum secure KEM such as Frodo ([BCD+16], updated in the round 3 of the NIST PQ competition [ABD+21]) can bring Post quantum security, which is not the case in the other works).

[12] We only consider way and back messages here, several messages in a row count as one.

The most recent work of Caforio, Durak and Vaudenay ([CDV21]) extends this vision by proposing an hybrid model to build an asynchronous ratcheted communication with additional data (ARCAD) from any strongly secure protocol (such as [JMM19], [DV19], or [PR18b]) that brings PCS, together with a lighter, symmetric cryptography based protocol that only enables PFS. For this latter part, they rely on the recent Encrypt-then Hash proposed by Yan and Vaudenay ([YV20]). They try to optimize the ratio efficiency/security by proposing an on-demand ratchet feature, that calls the most heavy protocol only when considered necessary: "The decision to ratchet or not could of course be made by the end user or rather triggered by the application at an upper layer".

3.3.6 Additional properties

Immediate decryption and out-of-order messages. Alwen, Coretti and Dodis introduced the notion of immediate decryption. The idea is to enable out-of-order messages and even to resist to a message loss. This directly reduces the PFS as older unused keys have to be kept in memory. This also affects the design. Acknowledgement mechanisms for instance, as the onion of [DV18] or the concatenation of [PR18b], are excluded: the update mechanism must not depend on previous messages. And Bob must be able to compute a non-received yet key from his state and an newer message, which supposes that this computation does not require randomness from Alice. Finally, in order to limit the PFS consequences, message keys shall be independent: from an old key preserved, one shall not be able to compute further keys (this banishes simple key hashing as PFS mechanism).

Recovery. Durak and Vaudenay wonder in [DV18] whether it is really a good thing to be able to recover from an impersonation. As for the discussion on correctness, even if this feature can be seen as practical at a first glance, it also says that one will not see that an impersonation has taken place. They introduce the RECOVER security notion. Briefly, in a protocol that is RECOVER secure, if Alice has been impersonated (meaning that Bob has received a trivial forgery) then Bob can no longer receive any message from the honest Alice. A solution to achieve RECOVER security is to provide a transcript dependant data within every message. This is done in [JS18b]: when Alice sends a message, she sends as additional data the hash of all the transcript she sent since the last update acknowledged by Bob. In [CDV21], the authors state that the protocol proposed in [PR18b] is not RECOVER secure. While [PR18b] does not include an history of the transcript in its messages, it is unclear how the RECOVER security can fail since the fine-grained PCS property ensures that Bob can not decrypt messages from the honest Alice after she has been impersonated and that their correctness definition demands that a fault in the decryption shall cut off the communication.

3.3.7 Conclusion

As described above, the security offered by the fine grained compromised model seems the most desirable. However, as we pointed out, this higher security requires impractical public key updatable primitives. Because of their update and acknowledgement mechanisms, the schemes proposed in [PR18b], [JS18b], [JMM19] and [DV19] see their efficiency dramatically impoverished if Alice and Bob do not regularly alternate their role of sender/receiver, which is quite in contradiction with the purpose of asynchronous messaging. On another hand, the flexibility offered by [CDV21] has to be handled carefully as this can easily lead to a huge difference between the security claimed by an application and the reality.

From Single to Multi-Device Instant Secure Messaging 4

The development of smartphones has made of secure instant messaging an essential everyday-life application. In parallel, more and more people started using several devices - a smartphone, a tablet or a laptop - to communicate. They wish to be able to frequently and rapidly switch between them. Security protocols such as SIM have to be adapted to this ever-changing multi-device setting. However, the modifications have to be as light as possible for the users and efficient so that it will be the same if we use this or that device. This chapter mainly contains the contributions that were published in [CDDF20], and in the corresponding full version [CDDF19].

Contents

As studied in chapter 3, an interesting property of ratcheted key exchange is that the confidentiality of past and future messages is still guaranteed even after an exposure of the long-term keys or of the state secrets. Regrettably, the Double Ratchet algorithm has been designed for device to device interaction and its use in a multi device context is more difficult. This was already underlined in the work of Unger *et al.* ([UDB+15]), which concludes about TextSecure that it introduces problems for multi-device support, while defining multi-device as an important usability criteria[1]. Consequently, each SIM application has developed its own strategy to solve this problem. We detail below three of them, WhatsApp, Facebook Messenger and Signal. A more detailed panorama can be found in [DGGL21]. This recent systematization of knowledge explores a various number of multi-device solutions, ours included. The authors underline that, even now, the academic literature on multi device instant messaging is very sparse (they admit three papers of interest, the classical but not up-to-date SoK of Unger *et al.* [UDB+15], a work of Atwater and Hengartner on the problematic of key distribution among multiple device using threshold cryptography [AH16] and our contribution), and their work is mostly based on white papers of existing Instant Messaging application.

4.1 Existing solutions

WhatsApp. The most widely used SIM is designed to be used on a single phone. However, in order to enable its users to communicate from a computer, WhatsApp developers released WhatsApp web. This interface establishes a secure channel between the "master phone" and the computer, with the former just pushing data from the server to the latter, and conversely. Thus, a user can use WhatsApp from its computer only if his phone is also connected[2].

Facebook Messenger. This SIM enables end-to-end encryption as an option, called secret conversation. A technical white paper issued in May 2017 [Fac17] explains that "Secret conversations with more than two devices use the Signal Protocol's group Messaging Protocol". In this solution, called Sender's Key, each device sends to the others (through a Signal Channel unused afterward) a same symmetric key: the Sender's key. This key is ratcheted through a key derivation function (KDF), without additional key exchange information. This protocol does not achieve future secrecy and does not offer the security we are looking for.

Signal. In April 2017, Open Whisper Systems (the company who developed Signal) released Sesame, a new protocol dedicated to multi-device secure messaging [MP17]. Sesame consists in establishing Signal sessions between all devices, as shown in Figure 4.2. It represents $\binom{n_A}{2} + \binom{n_B}{2} + n_A \cdot n_B$ Signal channels. If Alice has n_A devices and Bob n_B, it requires for Alice $(n_A - 1) + n_B$ encryptions for each message she sends, and as many ratchet executions. Adding or removing a device from a user's pool of devices is possible through opening/closing the corresponding pairwise channel[3]. In Sesame as in Facebook Messenger, Alice knows that Bob communicates from several

[1] It is interesting to note that the only solutions for conversation security that were identified to support multi device were either theoretical such as the puncturable encryption of Green and Miers ([GM15] or the IBE based protocol of Canetti, Halevi and Katz ([CHK03]) or already deployed protocols not designed for massive secure messaging, such as TLS or openPGP.

[2] While writing this manuscript, it was announced that WhatsApp multi device support was to come "soon" (see for instance https://www.theverge.com/2021/6/3/22466425/).

[3] Sesame offers two options: either each device owns private and public long term keys, or all devices of a single user share a unique private/public long term key pair. This key is used to identify new devices as belonging to the same user and avoid physical security checks. This second option (implemented by Signal) forces the Signal server to store

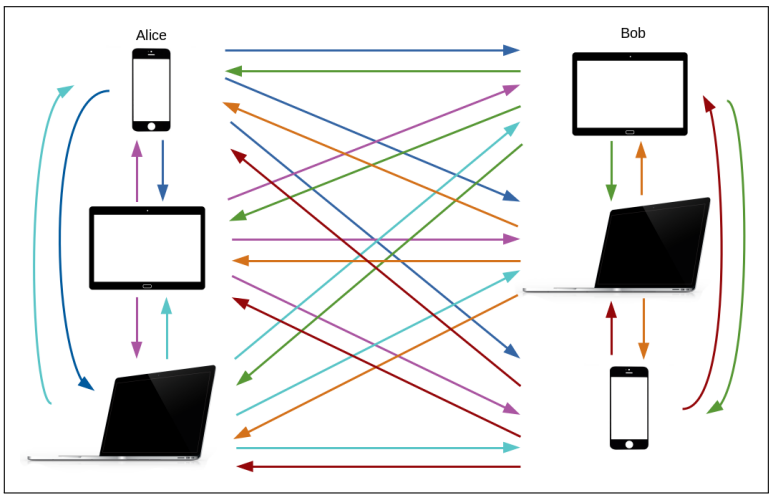


Figure 4.1 – Facebook's senders key version of the multi-device. Each arrow corresponds to a temporary pairwise channel used to transmit the device symmetric sender's keys to the others.

devices. She can even identify which channel - hence which device - sent the message.

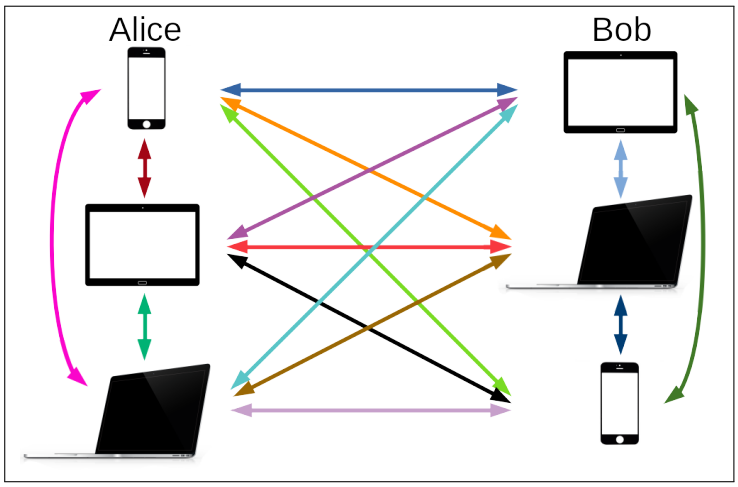


Figure 4.2 – Sesame multi-device protocol. Each array corresponds to pairwise Signal channel.

Messaging Layer Security. In a related area, Cohn-Gordon *et al.* proposed in [CCG+18] a solution for groups based on Diffie-Hellman trees. We describe this protocol more deeply in chapter 5. This solution could be adapted to the multi-device context, by considering each device as a single user. However, secure group messaging tries to tackle a broader and more complicated problem than secure multi-device messaging. We detail below some particularities to multi-device messaging that our contribution takes advantage of. More generally, we believe that designing a solution for the multi-device case is interesting given the evolution of users' practices, and that such a solution,

a "mail box" for each device of the same user. It also presents as major drawback the necessity of uninstalling and then installing again Signal's application on every device as soon as one is compromised.

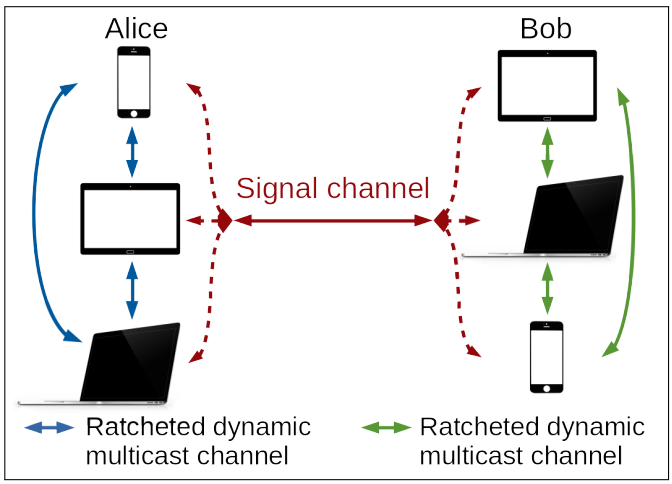


Figure 4.3 – Our Multi-Device Dynamic Ratcheted Key Exchange protocol. Only one Signal channel is needed between Alice and Bob.

besides being secure, must also be efficient and easy to use in order to be widespread.

Multi-device messaging vs. group messaging. In multi-device messaging, a single user owns and controls the different devices, while in group messaging, multiple users discuss using a single device each. Passive authentication is therefore easier to achieve in the multi-device case: received messages need to be authenticated as coming from a valid device but the identity of the sending device does not need to be revealed - the owner of the devices knows this information. Moreover, to authenticate a new device to another one of the same user, one can easily assume the devices will be physically close at some point. This means that a QR code can be used to exchange data between them (as it is the case in Sesame). Finally, assuming average usage, we do not take into account concurrent actions, such as revoking one's phone from one's tablet and conversely, at the same time. This also excludes the case when one honest device and a malicious one try to revoke each other at the same time. This could be handled at an application level by requiring a password or some personal data before revoking.

4.2 Our protocol overview

As a contribution we propose a multi-device protocol based on the classical two users Signal. In our solution, one user does not need to know how many devices the other has. Neither can he find which device his correspondent uses. This is an improvement in terms of privacy, as, for instance, the use of a particular device can leak information about your location. The idea is to open a specific multicast channel between a user's devices to broadcast the one Signal secret essential to perform the protocol: the ratchet secret key (rsk_A in subsection 3.1.2). As illustrated in Figure 4.3, each time one device of Alice sends a Signal message to Bob, it also sends a specific message to Alice other devices, containing the new Signal ratchet secret key. Thanks to this non interactive synchronization, all Alice devices have the same voice in the Signal conversation: they speak through the same Signal channel to Bob. On the way back, when Bob answers Alice through the unique Signal channel, his message is duplicated by the Server to all of Alice devices. A multicast channel is created for each Signal's session. To keep the security properties offered by the two-users ratchet, the multicast must

guarantee these properties.

A specific multicast for the devices. We propose as a first step a new primitive: a **Ratcheted dynamic multicast (RDM).** As for a traditional multicast, our RDM establishes a secure channel shared between several participants (in our case devices). It is dynamic since one can add or revoke devices during the execution of the protocol. The novelty is that the keys used to secure this channel are regularly updated, so as to obtain the forward secrecy and healing properties. This is the ratchet feature. The update can be done independently by any party. It is of utmost importance that each device remains independent in its ratcheting process, because in real life, one does not want to wait for all - or even a small part - of its devices to interact together before sending a message. For a similar reason, it is essential that our RDM is decentralized, as we want to avoid having a master device that one cannot afford to lose, have corrupted, or run out of battery. We also exclude a threshold mechanism, that would, as in [AH16], require several devices to be logged in (the Shatter protocol proposed in [AH16] does not require Alice to have several devices with her to communicate, however, if she only has her smartphone, her computer shall not be turned off and configured to answer automatically to communication requests).

As a contribution, we propose a security model, described in subsection 4.3.2 for this new primitive, as well as a construction detailed in subsection 4.3.3. We prove our construction secure in section 4.3.3. Our construction is based on standard well known primitives: an authenticated asymmetric encryption and a MAC scheme, as they were described in section 2.3.

Integrating our multicast with Signal. In a second step, we instantiate the integration of our multicast with Signal, to obtain **a Multi-Device version of Signal**.

Figure 4.4 represents a high level view of our solution, with Alice sending messages to Bob. Alice sends messages from any devices and Bob receives them on all of its devices. The square box numbers highlight some of our design particularities, that we motivate hereafter. We consider the sending device is $d_{A,i}$.

1. When Alice sends a message from any of her devices, this message is identically duplicated by the server and distributed to each of Bob's device. This can be done through mailboxes handled by the Server, who needs to know about Bob's devices (or at least about their numbers). This mailbox system is already offered by the Sesame solution in [MP17].

2. When the other devices receive a message corresponding to a symmetric Signal step performed by $d_{A,i}$, they have to perform the symmetric ratchet on their own to maintain their chain key up-to-date. (The corresponding ratchet secret is always transmitted to face up with undelivered messages. This is an implementation feature and could be omitted in a first simpler design).

3. When the other devices receive a message corresponding to an asymmetric Signal step performed by $d_{A,i}$, they receive the corresponding ratchet secret key rsk_A. From this key, they can perform the asymmetric ratchet, to derive the needed keys and maintain their state up-to-date.

4. As devices now share the ratchet secret, we need to change this secret when a device is revoked. A revocation hence induces an extra ratchet in the Signal conversation between Alice and Bob. As we do not want a newcomer to be able to read past messages, an additional ratchet also comes up with the joining process. Bob needs to know about this ratchet, otherwise the

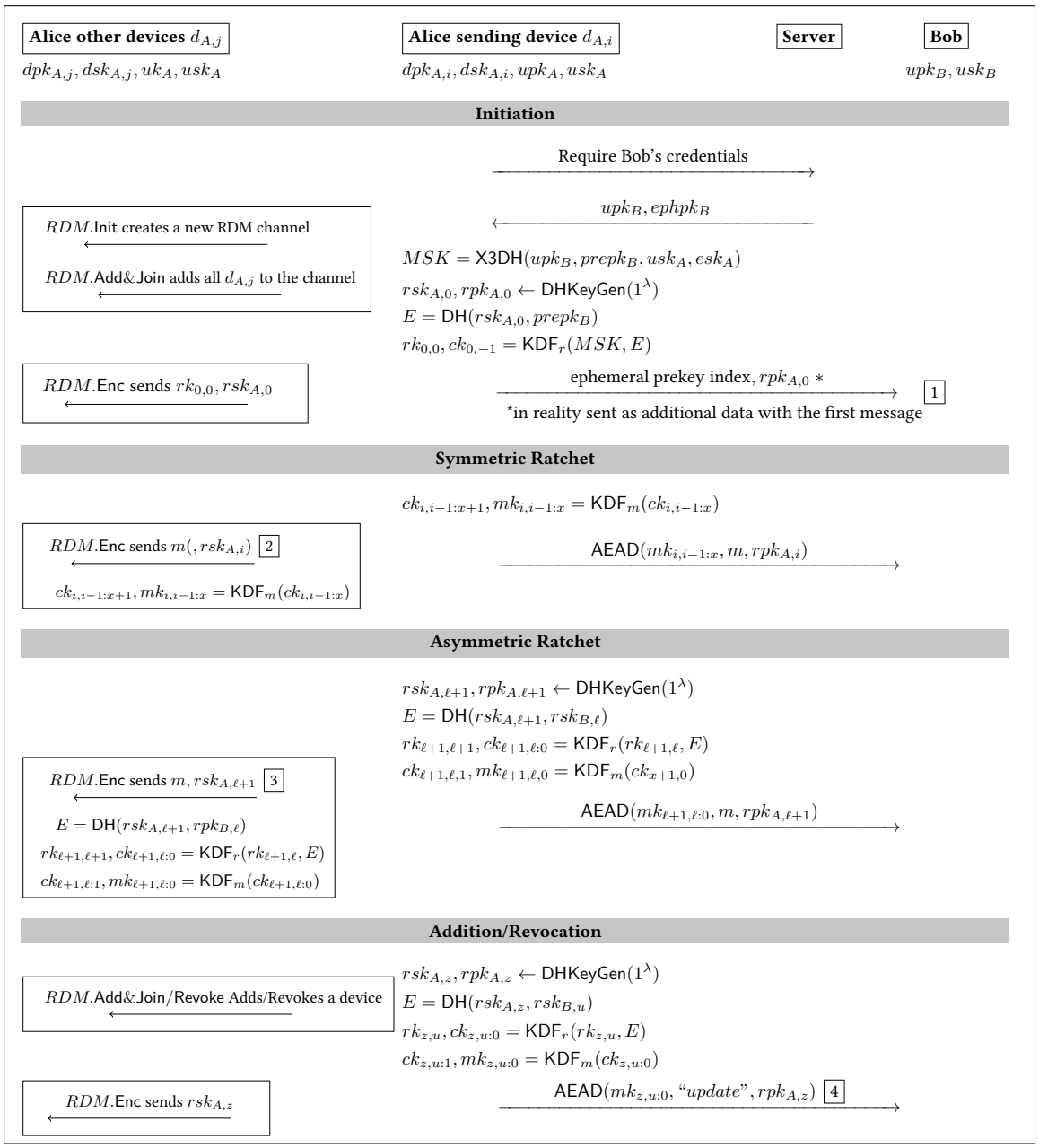


Figure 4.4 – Multi-Device Signal protocol. Signal procedures $\mathsf{KDF}_r, \mathsf{KDF}_m, \mathsf{X3DH}$, are defined in subsection 3.1.2. AEAD is defined in subsection 2.3.4. Boxed messages are sent between Alice devices. Without them, the figure represents Signal. Boxed numbers 1 to 4 are justified in section 4.2.

next message he sends would correspond to old keys, that the revoked device knows. An update message is sent to Bob, to let him know about the ratchet. Bob knows there has been a ratchet, but he can not know if it corresponds to an addition, a revocation, or a security update, and he has no clues about which devices are concerned. We take those extra ratchets into account in our security analysis.

We detail the above description in subsection 4.3.3. We introduce in subsection 4.3.2 a security model for such a RDM primitive. We introduce some important definitions and we detail how the freshness conditions in Signal's model need to be updated to take into account the multi-device feature, in particular the dynamic aspects. Our multi device messaging model is based on the multi-stage model of [CCD+17] recalled in subsection 3.2.2. However, one could plug our RDM on another RKE security model, with the same adaptations on the freshness conditions, obtaining a flavor of Multi-Device Ratcheted Key Exchange (MDRKE). We implement our solution over the Signal library libsignal-protocol-java accessible on Signal GitHub account. We give details and results in section 4.5.

How do we deviate from Signal. One of our goal is to upgrade the existing Signal protocol in a transparent way. However, one modification was unavoidable: the introduction of a device key, that every device generates for itself, before registering to the Signal server. This key is used to initiate the RDM channels between devices, and to add a new device. This key also plays a main part during the revocation process. In this precise case, we allow the renewal of the Signal ephemeral keys ($ephpk, ephsk$, that gathers the mid-term keys and one-time pre keys in the description of subsection 3.1.2) and user keys (upk, usk, that corresponds to the long term identity keys in subsection 3.1.2). In the original Signal, the user key cannot be modified without unregistering then registering again and thus closing all current conversations. In our solution, the server accepts a new user key for Alice if it is authenticated with one of Alice's device key. On Bob's side, this will be exactly as if Alice had registered a new account (as it is now in Signal). The main advantage is for Alice to keep her current conversations when revoking a device. If she had registered again, she also would have to add her devices again. Another deviation from the original Signal is that we make it possible to achieve several ratchets in a row on Alice's side (instead of the ping-pong pattern adopted by the original Double Ratchet). We show that this has no consequence on the security, nor on the possibility to deal with out-of-order messages. However, it implied for us a small patch in the Signal library as explained in section 4.5.

Our choices vs. Signal's Sesame solution. Our solution differs from Sesame or Facebook solutions in that, in our construction, a user is ignorant about his correspondent's devices. A message sent by Bob is only encrypted once for Alice, instead of being encrypted for each device of Alice. This message is also encrypted only once for all of Bob's other devices, instead of once per device. The server will be in charge of broadcasting the message to the appropriate devices. The authentication of a new device is also different. The Sesame protocol offers two options: the first requires all the devices of Alice to share a common IDkey. When a new device is added, it obtains this IDkey. Bob recognizes the new device as a device of Alice since it has the same IDkey. This makes the IDkey a very sensitive data. In [CCG16], the authors clearly stipulate that this feature prevented the TextSecure messaging app from achieving post-compromise security. Concretely, if an attacker learns Bob's ID key, he can further register his own device as Bob's one and enter any session, even after a ratchet. The ratcheting mechanism does not provide PCS any more. This attack has recently been further developed in [WBPE21] (to be published at DIMVA 21). In this paper, the author also show that our contribution is not weakened by this attack. In the second option, the devices do not share a common key. When Alice adds a device, Bob should physically authenticate this device to be sure it is honest and belongs to Alice. In our solution, we only require a new device of Alice to be authenticated by another device of hers.

4.3 A Ratcheted Dynamic Multicast as a new primitive.

As a first contribution, we introduce a new protocol for multicast communication. The idea behind the ratchet feature is that the protocol is stateful and the state evolves during the execution of the protocol. The goal is to strengthen the security of the channel. In the security model, it means that the adversary can be given more abilities than in a non-ratcheted version. As we explained in subsection 2.3.1, the traditional broadcast solutions do not fit our use case as they either require a center or interactivity between the users. We are not either in a classical multi-recipient scheme as we require a group notion which implies a form of authentication between the participants. From then, we will consider participants in the RDM as devices.

4.3.1 A RDM definition

We start by giving a formal description of a RDM. Each device i maintains two states. The device state, π_i, is valid for all the sessions of the protocol. It registers long-term private key and public key: $\pi_i.sk$, $\pi_i.pk$. The session state π_i^s is valid only for the session s of the protocol. It contains the following information:

- $rand$, the ephemeral information of the state.
- $devices$, the public keys of all devices involved in the session.
- PK, the current session public key for the group $\pi_i^s.devices$.

Protocol description. A Ratcheted dynamic multicast (RDM) is defined by nine algorithms:

- $\mathsf{SetUp}(1^\lambda, i) \to \pi_i$. Generates secret and public keys (sk_i, pk_i) and creates a device state π_i.
- $\mathsf{Init}(\pi_i, s) \to \pi_i^s$. Initiates a new session s of the protocol. Generates a session state π_i^s for this session.
- $\mathsf{Enc}(m, \pi_i^s) \to C_{enc}, \pi_i^s$. On input a message m and a session state π_i^s, returns a ciphertext C_{enc} and the updated state π_i^s.
- $\mathsf{Dec}(C_{enc}, \pi_j^r) \to m, \pi_j^r$. On input a ciphertext C_{enc} and a session state π_j^r, returns a message m and the updated state π_j^r.
- $\mathsf{Add\&Join}(\{pk_{j_\ell}\}_{\ell\in[1,z]}, \pi_i^s) \to C_{add}, C_{join}, \pi_i^s$. On input a set of public keys $\{pk_{j_\ell}\}_{\ell\in[1,z]}$ and a session state π_i^s (of the device that adds), returns information C_{join} for the new devices, C_{add} for the already enrolled devices and the updated state π_i^s.
- $\mathsf{DecJoin}(C_{join}, \pi_j, r) \to \pi_j^r$. On input a ciphertext C_{join}, a device state π_j, and a session identifier r, returns a new session state π_j^r.
- $\mathsf{DecAdd}(C_{add}, \pi_k^o) \to \pi_k^o$. On input a ciphertext C_{add} and a session state π_k^o, returns the updated session state π_k^o.
- $\mathsf{Revoke}(pk, \pi_i^s) \to C_{rev}, \pi_i^s$. On input a public key pk and a session state π_i^s, returns a ciphertext C_{rev} and the updated state π_i^s.
- $\mathsf{DecRevoke}(C_{rev}, \pi_k^o) \to \pi_k^o$. On input a ciphertext C_{rev} and a session state π_k^o, returns the updated state π_k^o.

Implicitly, in all decryption algorithm, if the decryption fails (returns an error symbol $\perp$), then the state is maintained. Our correctness definition will then correspond to the Corr described in subsection 3.3.2.

Towards a more general description. What is expected from a primitive description is a high level view, with the minimal input and output interfaces, that leaves a great freedom to a protocol designer. The algorithms described above may seem to direct the construction too much. We give arguments for why we chose such a level of detail. Firstly, it is important to notice that, due to the addition and revocation feature, a dynamic multicast is to be analysed as a complex protocol more than as a mere encryption scheme. It requires messages to be exchanged between participants. This is the sense of the Init algorithm, that opens a session of the protocol. This may be compared to the Execute given in [PPS12]. This Execute represents the initial protocol between the users that leads to a broadcast group definition. However, in the model of [PPS12], the adversary is only given a single access to the Execute oracle. As we will see in subsection 4.3.2, we chose to analyse our multicast in a concurrent session setting.

Concerning a multicast encryption, as for an encryption scheme, Enc and Dec are the minimal functionalities one can expect. As our protocol is statefull, there is no surprise that the state comes up the algorithms definition. Then the last question is: how generic can the dynamic part be ? Add and Revoke are necessary to obtain a fully dynamic protocol. In [PPS12] again, the author do not detail their Join algorithm but only describe it as an interactive protocol between the group and the newcomer. We could have, in a similar manner, gathered our Add&Join, DecJoin and DecAdd in a single Add procedure for instance (and the same way, we could have included the DecRevoke oracle together with the Revoke describing it as a procedure between the enrolled devices). However, as in our primitive, the addition and revocation mechanisms are not necessarily interactive, it seemed less intuitive to gather the whole procedure in a single call. The choice to keep separate the initiation of the addition procedure by a single member and the finalisation by the rest of the group on the one side and the newcomer on the other side seems even more legit when one thinks of an adversary. In the security model, the adversary has access to the different algorithms that compose the protocol. If the addition was to be considered as a sub protocol, then how do $\mathcal{A}$ could interact with it ? In a real life execution, a strong adversary may tamper with the messages that concern the addition or revocation in the same way as with the more basic encryption and decryption. This is why the nine algorithms that compose the RDM description are necessary to represent the functionalities of this primitive.

4.3.2 An appropriate security model

We first give an intuition of the security expected from a RDM primitive. We expect a RDM to provide indistinguishability under chosen-ciphertext attacks, as defined in subsection 2.3.1, as well as forward secrecy and healing. We define our security model by starting from an ideal case where the adversary has full powers, and then excluding the attacks that we consider as unavoidable. The adversary controls the execution of s sessions of the protocol and he can obtain all the secret information he wishes. At some point, he can query an indistinguishability challenge on one session. He then has to distinguish between a real ciphertext honestly produced by this session or some randomness (we adopt the real-or-random modelisation for indistinguishability). We exclude the cases where he could trivially win, or the attacks that we consider as unavoidable by defining some freshness conditions, that are formally defined in section 4.3.2.

We introduce three necessary definitions. Firstly, we formalize the notion of step of a protocol. A session can live for a long life time (weeks, months) and some secret data may evolve during this period. Steps are meant to follow this evolution. They are our multicast equivalent to the Epoch defined for the RKE[4] (*cf.* subsection 3.3.2). Secondly, we define matching sessions, based

[4]We did not choose the term Epoch because, if we assemble our RDM with a RKE, the Epochs of the RKE and the steps

on [BR94]. This is necessary because we consider a multi-session context. Our definition helps us to define the correctness of our protocol: if two participants are involved in a same execution and have reached corresponding steps - *i.e.* if they are matching, they should be able to communicate together. Moreover, as matching sessions may share common secret data, the adversary's powers are also defined "matching-session" wise. Finally, because of the dynamic feature, several sessions that correspond to a same execution of the RDM may not be present at the same time, and so do not match. They are however related. We introduce the notion of chained sessions, to take this relationship into account.

Let $\{\mathtt{d}_1, \ldots, \mathtt{d}_{n_d}\}$ be the devices participating in the protocol. Each device $\mathtt{d}_i$ is modelled by an oracle π_i and each session s executed by a device $\mathtt{d}_i$ (session (i, s)) is modelled by an oracle π_i^s. We identify a session s by the number s_i of sessions already run by i. Oracles maintain states as defined in subsection 4.3.1. In the following, we assimilate the oracles and their state.

Protocol steps. Data registered in a device state for a session will change during the execution of the protocol. To model this phenomenon, we consider steps of the protocol. Each Enc, Add&Join, Revoke or corresponding decryption algorithm advances the protocol to a new step. Steps are formalized through a counter t, set to 0 at initiation and incremented by oracle queries. This counter is included in the oracle session state with $\pi_i^s.\mathtt{step}$. It is not necessary in an implementation but needed by the model. (In a general way, we use the `typewriter` typo for model specific elements). Going from one step to another indicates that the algorithm has processed without error. Intuitively, steps will embody the healing and forward secrecy properties: some restrictions can be needed at some step t and released at step $t + 1$ (or reversely), meaning that the confidence is back (is still there for past steps). We refer with (i, s, t) to the session (i, s) at step t. We note $\pi_i^s[t]$ when we refer to oracle's state π_i^s as it was at step t. We note $\pi_i^s[t].\mathtt{X}$ the access to item $\mathtt{X}$ at step t.

Matching sessions. We denote $\pi_i^s.\mathtt{sid}$ the transcript of the protocol executed in session (i, s), that is, the concatenation of all messages C_i sent or received by π_i^s. We write $\pi_i^s[t_s].\mathtt{sid} = C_i[0]\|C_i[1]\|\ldots\|C_i[t_s]$. As no message is sent or received for the initiation, the first component of a $\mathtt{sid}$ for a session running the Init procedure is set to INIT. We refer to a session created by an Init algorithm as an initial session. As all devices are playing similar roles, we do not consider roles in our definition of the matching sessions. Since devices can join and leave during the protocol, we define a matching that is step-wise.

Definition 4.1 (Matching sessions at some step.)**.** *One says (i, s, t_s) and (j, r, t_r), $t_s \geq t_r$ ((i, s) joined first), are matching if $\exists\ \mathtt{sid}'$ substring of $\pi_i^s[t_s].\mathtt{sid}$ such that $\pi_i^s[t_s].\mathtt{sid} \tilde{=} \mathtt{sid}'\|\pi_j^r[t_r].\mathtt{sid}$ ($\mathtt{sid}'$ eventually empty). The symbol $\tilde{=}$ stands for the following definition: $\pi_i^s[t_s].\mathtt{sid} \tilde{=} \mathtt{sid}'\|\pi_j^r[t_r].\mathtt{sid}$ if, $\forall t \in [0; t_r]$:*

- *either $t > 0$ and $C_i[t_s - t_r + t] = C_j[t]$,*
- *either $t > 0$ and $C_i[t_s - t_r + t] = (C_{add}, C_{join})$ or C_{add} and $C_j[t] = C_{add}$,*
- *either $t > 0$ and $C_j[t] = (C_{add}, C_{join})$ or C_{add} and $C_i[t_s - t_r + t] = C_{add}$,*
- *or $t = 0$ and $C_i[t_s - t_r] = (C_{add}, C_{join})$ or C_{add} and $C_j[0] = C_{join}$ with (C_{add}, C_{join}) having been produced by the same Add&Join call.*

of the RDM would complete each other but not necessarily correspond.

As devices can join the protocol at any moment, we define a way to link sessions that corresponds to a same execution but were not present at the same time. This composes chains of sessions, as illustrated in Figure 4.5.

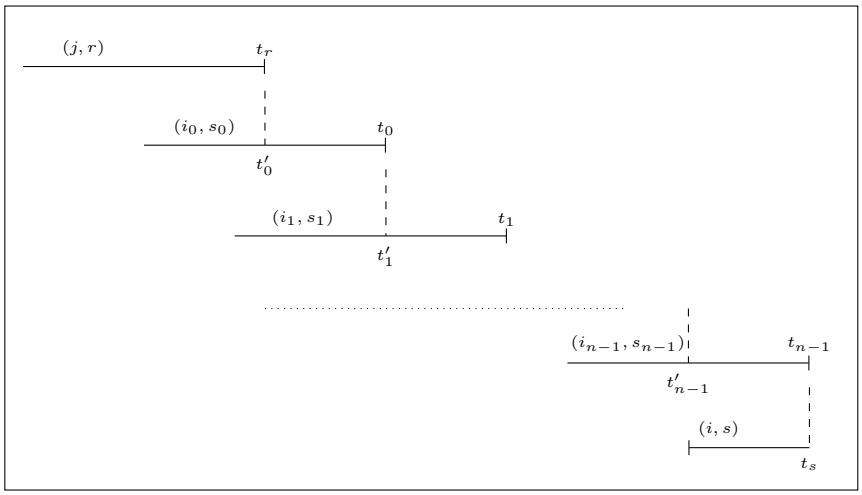


Figure 4.5 – A chain of sessions between (i, s, t_s) and (j, r, t_r).

Definition 4.2 (Chained sessions.). *A session (j, r, t_r) is chained with (i, s, t_s) if t_r is maximal and there exists n sessions (i_α, s_α), and n couples (t'_α, t_α), $t'_\alpha \leq t_\alpha$, $\alpha \in [0, n-1]$ such that:*

- *(j, r, t_r) and (i_0, s_0, t'_0) are matching,*
- *$\forall \alpha \in [0, n-2]$, $(i_\alpha, s_\alpha, t_\alpha)$ and $(i_{\alpha+1}, s_{\alpha+1}, t'_{\alpha+1})$ are matching,*
- *$(i_{n-1}, s_{n-1}, t_{n-1})$ and (i, s, t_s) are matching.*

$\{(i_\alpha, s_\alpha, t_\alpha)\}_{\alpha \in [0,n-1]}$ is called a chain of sessions between (i, s, t_s) and (j, r, t_r).

Note that the above definition is "one-way" : a session is chained with a more advanced session, in the sense, with a session that have processed more recent messages.

We now come to the definition of the correctness. We expect matching sessions to be synchronised and, as a consequence, to be able to communicate together. We formalise this statement in the following definition.

Definition 4.3 (Correctness.). *Suppose a passive adversary that sees communications and may only disturb their delivery. A* RDM *is said to be correct if, for all matching sessions (i, s, t_s) and (j, r, t_r), for all messages m,*

$$\mathsf{Dec}(\mathsf{Enc}(m, \pi_i^s[t_s]), \pi_j^r[t_r])) = m.$$

In the following, we consider the security of a multicast when facing a more powerful adversary. The goal is to ensure that the scheme provides the expected confidentiality on the messages.

The adversary's powers. As in the original IND-CCA experiment, the adversary $\mathcal{A}$ can query for one Challenge of indistinguishability. He is given access to oracles that enables him to perform the whole protocol: OInit, OEnc, ODec, OAdd&Join, ODecAdd, ODecJoin, ORevoke, ODecRevoke. The oracle OInit defines the experiment in a multi-session context. In the original definition of IND-CCA, giving the adversary the possibility to corrupt the key would be a non sense, as only one

long term key is considered. In a RDM protocol, one expects the ratchet feature to bring PFS and PCS. Hence, the adversary can query a RevealLongTerm on a device to obtain its long term secret key, and he can choose to query RevealState[5] to obtain the state secrets of any device. We consider as state secrets values that needs to be memorized from one message ton another for instance. Randomness used for encryption is not considered as part of the state. Finally, we do not consider out-of-order messages, messages are supposed to be transmitted in the order they were created.

We complete the state description with flags, not required for the implementation, to keep trace of the adversary's queries:

- `revLT`. It is a flag on global state. It is set to $false$ at Register, set to $true$ whenever RevealLongTerm is called, if the device is not active (see below).

All the followings are session state flags:

- `revState`. Set to $false$ at Init, set to $true$ whenever RevealState is called on this session.
- `challenge`. Set to $false$ at Init, set to $true$ whenever Challenge is called on this session or on a matching session.
- `active`. Set to $true$ if the device i has been called by oracle OEnc in session s. A device gets active in the protocol as soon as it sends a message.

We note $\pi_i^s.\texttt{flag}$ for $\pi_i^s.\texttt{flag} = true$ and $\neg\pi_i^s.\texttt{flag}$ for $\pi_i^s.\texttt{flag} = false$.

The security model description We give a pseudo code description of the model in Figure 4.6, gathering all the oracles accessible to the adversary.

Freshness. The natural restrictions defined here are meant to exclude unavoidable attacks or cases where the adversary could win trivially. These restrictions are often valid for a session and all the corresponding chained sessions (not only matching sessions). This expresses the fact that a device has to participate regularly to the protocol to update its state. This is inherent to the ratchet process: the participants have to be actively involved for the ratchet to be operational. One of the direct consequence of this remark, is that we consider that the session specific data are equal to the long term data until a participant is active. This gives the adversary two ways of accessing long-term data, `revLT` and `revState`. We carefully take into account these two paths for the adversary to trivially win the Game. We consider the freshness condition directly in the experiment description. A main reason is to make easier the composition with the messaging part in a second step. In order to formalise the freshness conditions in the pseudocode representation of the model, we introduce the following notation:

`NoRevState-NoInactiveRevLT(i, s, t)` =

- $\neg[\pi_i^s[t].\texttt{revState} \vee (\pi_i.\texttt{revLT} \wedge \neg\pi_i^s[t].\texttt{active})]$ and
- $\forall(j, r, t_r)$ chained with (i, s, t) and non revoked, $\neg[\pi_j^r[t_r].\texttt{revState} \vee (\pi_j.\texttt{revLT} \wedge \neg\pi_j^r[t_r].\texttt{active})]$.

[5] Again, about the vocabulary, our RevealState is equivalent in spirit to the Expose introduced for the RKE. However, as the exposure was introduced in a context where there are no long term key queries, it seemed to us more relevant to keep the traditional RevealState/RevealLongTerm vocabulary.

$\mathsf{Exp}^{\text{RDM-IND}}_{RDM,n_d,q,\mathcal{A}}(n)$

$b \leftarrow_\$ \{0,1\}$
$P \leftarrow \bot, initialdata \leftarrow \bot, C* \leftarrow \bot$
for $i = 1, \ldots, n_d$ **do**
  $\pi_i \leftarrow \mathsf{SetUp}(1^\lambda, i), s_i \leftarrow 0$
  $initialdata \leftarrow initialdata \cup \{\pi_i.pk\}$
$b' \leftarrow \mathcal{A}^{Oracles,Challenge}(initialdata)$
return $b = b'$

OInit(i)

$s_i \leftarrow s_i + 1, s \leftarrow s_i$
$\pi_i^s[0] \leftarrow \mathsf{Init}(\pi_i, s)$
$\pi_i^s.\texttt{step} \leftarrow 0, E_i^s \leftarrow 0, V_i^s \leftarrow \emptyset$
return

OEnc(m, i, s)

$t \leftarrow \pi_i^s.\texttt{step}$
$C_{enc}, \pi_i^s[t+1] \leftarrow \mathsf{Enc}(m, \pi_i^s[t])$
$V_i^s \leftarrow V_i^s \cup C_{enc}$
for $T \geq t+1$ **do**
  $\pi_i^s[T].\texttt{revState} \leftarrow false$
  $\pi_i^s[T].\texttt{challenge} \leftarrow false$
  $\pi_i^s[T].\texttt{active} \leftarrow true$
$E_i^s \leftarrow t+1, \pi_i^s.\texttt{step} \leftarrow t+1$
return C_{enc}

ODec(C_{enc}, j, r)

$t \leftarrow \pi_j^r.\texttt{step}$
Req. $\neg(C_{enc} = C^* \wedge \pi_j^r[t].\texttt{challenge})$
Req. NoRevState-NoInactiveRevLT(j, r, t)
$\vee\, \exists(k, o, t_o)$ matching (j, r, t) such that $C_{enc} \in V_k^o$
$m, \pi_j^r[t+1] \leftarrow \mathsf{Dec}(C_{enc}, \pi_j^r[t])$
$\pi_j^r.\texttt{step} \leftarrow t+1$
return m

OAdd&Join($\{j_\ell\}_{\ell \in [1,z]}, i, s$)

$t \leftarrow \pi_i^s.\texttt{step}$
$C_{join}, C_{add}, \pi_i^s[t+1] \leftarrow \mathsf{Add\&Join}(\{\pi_{j_\ell}.pk\}_{\ell \in [1,z]}, \pi_i^s[t])$
if $\exists\ \ell$ such that $\pi_{j_\ell}.\texttt{revLT}$ **then**
  for $T \geq t$ **do** $\pi_i^s[T].\texttt{revState} \leftarrow true$
$V_i^s \leftarrow V_i^s \cup \{C_{add}\}, P \leftarrow P \cup \{C_{join}\}$
$\pi_i^s.\texttt{step} \leftarrow t+1$
return C_{join}, C_{add}

ODecJoin(C_{join}, j)

Req. $C_{join} \in P$
$s_j \leftarrow s_j + 1, r \leftarrow s_j$
$\pi_j^r[0] \leftarrow \mathsf{DecJoin}(C_{join}, \pi_j, r)$
$\pi_j^r.\texttt{step} \leftarrow 0, V_j^r \leftarrow \bot, E_i^s \leftarrow 0$
return

ODecAdd(C_{add}, k, o)

$t \leftarrow \pi_k^o.\texttt{step}$
Req. NoRevState-NoInactiveRevLT(k, o, t)
$\vee\, \exists(j, r, t_r)$ matching (k, o, t) such that $C_{add} \in V_j^r$
$\pi_k^o[t+1] \leftarrow \mathsf{DecAdd}(C_{add}, \pi_k^o[t])$
$\pi_k^o.\texttt{step} \leftarrow t+1$
return

ORevoke(pk, i, s)

$t \leftarrow \pi_i^s.\texttt{step}$
$C_{rev}, \pi_i^s[t+1] \leftarrow \mathsf{Revoke}(pk, \pi_i^s[t])$
$V_i^s \leftarrow V_i^s \cup \{C_{rev}\}$
$\pi_i^s.\texttt{step} \leftarrow t+1$
return C_{rev}

ODecRevoke(C_{rev}, k, o)

$t \leftarrow \pi_k^o.\texttt{step}$
Req. NoRevState-NoInactiveRevLT(k, o, t)
$\vee\, \exists(j, r, t_r)$ matching (k, o, t) such that $C_{rev} \in V_j^r$
$\pi_k^o[t+1] \leftarrow \mathsf{DecRevoke}(C_{rev}, \pi_k^o[t])$
$\pi_k^o.\texttt{step} \leftarrow t+1$
return

RevealLongTerm(i)

Req. $\neg\exists s, t_s$ such that
$\pi_i^s[t_s].\texttt{challenge} \wedge \neg\pi_i^s[t_s].\texttt{active}$
$\pi_i.\texttt{revLT} \leftarrow true$
return sk_i

RevealState(i, s)

$t \leftarrow \pi_i^s.\texttt{step}$
Req. $\neg\pi_i^s[t].\texttt{challenge}$
for $T \geq E_i^s$ **do** $\pi_i^s[T].\texttt{revState} \leftarrow true$
if $\neg\pi_i^s.\texttt{active}$ **do** $\pi_i.\texttt{revLT} \leftarrow true$
return $\pi_i^s[t].rand$

Challenge(m_0, m_1, i, s)

$t \leftarrow \pi_i^s.\texttt{step}$
Req. NoRevState-NoInactiveRevLT(i, s, t)
$C^*, \pi_i^s[t+1] \leftarrow Enc(m_b, \pi_i^s[t])$
for $T \geq t$ **do**
  $\pi_i^s[T].\texttt{challenge} \leftarrow true$
  $\pi_i^s[T].\texttt{revState} \leftarrow false$
for (j, r, t_r) chained with (i, s, t) such that:
$\pi_j.pk \in \pi_i^s.devices$ **do**
  for $T \geq t_r$ **do**
    $\pi_j^r[T].\texttt{challenge} \leftarrow true$
$\pi_i^s.step \leftarrow t+1$
return C^*

Figure 4.6 – The full RDM-IND security game given in pseudo code. Req. stands for require.

By checking $\texttt{NoRevState} - \texttt{NoInactiveRevLT}(\texttt{i}, \texttt{s}, \texttt{t})$, we ensure that the state of(i,s) is not revealed, would it be by a RevealState or a RevealLongTerm while i is not active in that session or by a similar query on a chained session. We consider non revoked chain sessions here and not only matching session as, if a session is chained, it is waiting to process all the messages until the one that corresponds to step t of (i, s). As its state enables to process all the messages, a leakage of its state gives $\mathcal{A}$ access to (i, s, t)'s state also. This traduces that the devices share at least part of their state. Without the condition on chained session, the $\texttt{NoRevState} - \texttt{NoInactiveRevLT}(\texttt{i}, \texttt{s}, \texttt{t})$ condition implies that each devices maintain its own ratcheted channel.

We detail below the different freshness conditions that appear in the model. The first two restrictions exclude traditional trivial attacks. The third means forward secrecy can be achieved solely for active participants. The fourth models the healing property of the ratchet. The fifth excludes impersonation attacks and finally, the last point models an authentication procedure when adding a new participant, which we consider out of the scope of this protocol.

1. $\mathcal{A}$ shall not RevealState state secrets just before the challenge. This is prevented by line 2 of Challenge.
2. $\mathcal{A}$ shall not RevealState a device concerned with the challenge. This is prevented by line 2 in $RevealState$ and lines 7-9 in Challenge. The sequels of a RevealState are cancelled by a OEnc. Line 6 in OEnc ensures the security is back for i's secret after i performs an encryption. This ensures the security is back for $\texttt{d}_i$'s secret after $\texttt{d}_i$ performs an encryption. A RevealState on a device only threatens the steps from the last encryption and until the next. This corresponds to the healing property. It also means that forward secrecy depends on devices regularly sending messages.
3. $\mathcal{A}$ shall not RevealLongTerm a non active device before or after the challenge. This is prevented by line 2 of Challenge and lines 1-2 of RevealLongTerm. A device comes active as soon as it performs an encryption (it enters the ratcheting process).
4. $\mathcal{A}$ shall not RevealState and use the secret state information to maliciously send an encrypted message with its own new random, revoke someone, or join a non authorized corrupted device. There is nothing we can do against this kind of impersonation attack, and the two user ratchet is also vulnerable to it. We prevent this by introducing the register V_i^s that keeps track of ciphertexts produced by the OEnc, OAdd&Join and ORevoke oracles. Line 3 of Oracle ODec, and line 2 of ODecAdd and ODecRevoke check whether the incoming ciphertext has been produced legitimately. If not, those lines verify whether there was a RevealState at the same step.
5. Register P initialized in line 2 of the global experiment prevents $\mathcal{A}$ from joining a device in a non existing session or after an exposure. This models a physical authentication procedure between the device that adds and the new device.

Definition 4.4. *(Secure Ratcheted Dynamic Multicast) A* RDM *running with n_d devices is a secure Ratcheted Dynamic Multicast if it is correct and, for all adversaries $\mathcal{A}$, running in polynomial time, making at most q queries to the oracles, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\mathsf{Adv}^{\text{RDM-IND}}_{\mathcal{A},\mathsf{RDM},n_d,q}(\lambda) = \left| \Pr\left[\mathsf{Exp}^{\text{RDM-IND}}_{\mathcal{A},\mathsf{RDM},n_d,q}\right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)\,.$$

We denote $\epsilon_{\text{RDM-IND}}$ this advantage.

4.3.3 Our construction.

As a second contribution, we propose a construction that realise a Ratcheted Dynamic Multicast, and prove it secure in the model described in subsection 4.3.2. We first give in Figure 4.7 a high-level view of our construction. The detailed pseudo code description is given in Figure 4.8.

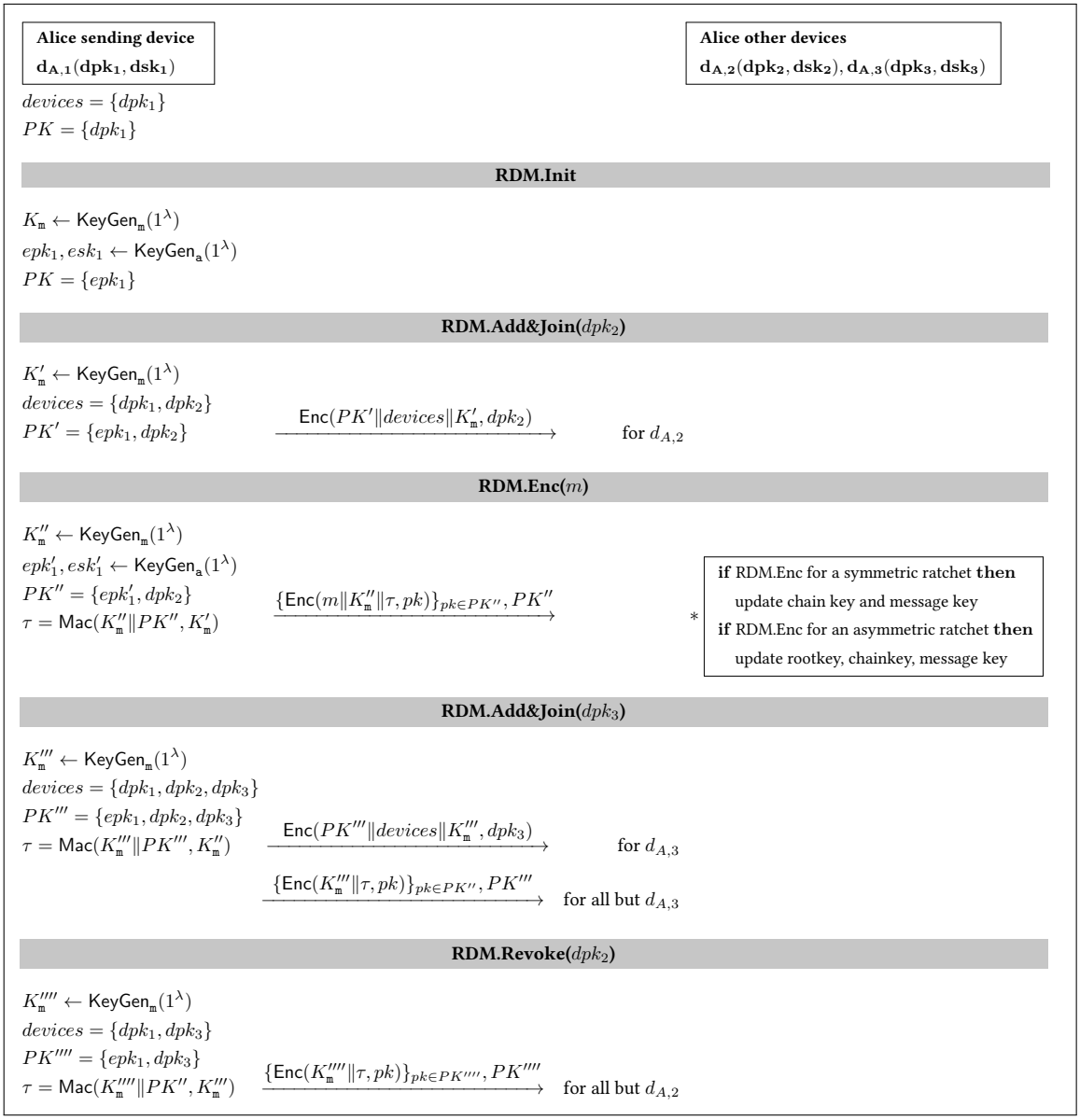


Figure 4.7 – Our RDM protocol. The sending device can change for each procedure. The algorithms (KeyGen_m, Mac, Verif) and (KeyGen_a, Enc, Dec) define a Mac (respectively an asymmetric encryption) as in section 2.3. The instructions * detail the integration of our RDM with Signal.

The main idea is that the keys used to encrypt the multicast messages are updated regularly. We base our solution on parallel asymmetric encryption, as studied in [BPS00] or [BBM00] and detailed in subsection 2.3.2. The authenticated asymmetric encryption scheme given by th algorithms

($\mathsf{KeyGen_a}$, Enc, Dec) is defined as in subsection 2.3.1, is used as a multicast in the naive way: PK is the set of public keys of all devices $\mathtt{d}_i$ concerned with the encryption. $\mathsf{mEnc}(m, PK)$ stands for $\{\mathsf{Enc}(m, pk)\}_{\forall pk \in PK}$. We consider that the number of devices remains reasonable: around ten for each user does not seem so restrictive. This design allows us to choose among well-known and proven secure primitives, as detailed in section 2.3.1.

As we have seen, broadcast encryption, as a primitive, does not fit our use case. This is why we introduced the notion of RDM. But one could try to construct a RDM using a decentralized broadcast scheme. However, the most efficient construction given in [PPS12] can not be used as a building block for a RDM, for it requires interactivity between the participants. More generally, broadcast encryption solutions offer fewer implementation evaluations.

An asymmetric ratchet. The Diffie-Hellman ratchet implemented in Signal is only possible for two users. With more than two parties, multiparty computation could be thought of as an option, but we do not want to wait for all, or even a minimum number of devices to be present before sending a message: each device has to be autonomous in its ratcheting process. Our ratchet consists in generating new ephemeral asymmetric keys epk, esk for the device which sends a message. The multicast public key is updated with the new ephemeral public key epk. Here, we take advantage of the multi-device context. As all the devices belong to a single person, we consider that no honest device will try to exclude another device maliciously (by mis-updating the multicast public key). When any device updates its ephemeral key pair, the others only receive the updated common public key. They do not need to know about who updated it. However, when a device wants to revoke another device, it has to know which ephemeral public key to erase from PK. We deal with this by considering there exists a correspondence between the list $devices$ of long term keys recorded in each device state and the list PK of ephemeral public keys. Requirements in the Add&Join algorithm prevent a device from being present in the group several times. In such a case, revoking this device once would not be enough to be sure it is definitively out of the protocol.

Passive authentication. Another important point is that the messages have to be identified as coming from an honest device, but again, its identity does not matter. Our solution provides passive authentication thanks to a Mac key $K_{\mathtt{m}}$ shared between the devices. A new Mac key is generated with every action: sending a message, joining, or revoking a device. Otherwise, an adversary who could access the Mac key at some step could impersonate any device at any step further. This new Mac key is authenticated under the previous one, creating an authentication chain. This solution is less expensive than generating new signature key pairs regularly.

The detailed construction. We give a pseudocode detailed description of our RDM protocol in Figure 4.8. The restrictions in the Add&Join procedure are there to prevent a same device to be added several times. The requirement in the Revoke procedure prevents a device to revoke itself. Ephemeral data esk, epk and $K_{\mathtt{m}}$ constitute the randomness $\pi_i^s.rand$ of the model. For readability reasons, we keep them separate in the construction and refer to them with $\pi_i^s.esk$, $\pi_i^s.epk$, and $\pi_i^s.K_{\mathtt{m}}$. To be able to initialize a session, a device must have processed a SetUp to generate its global state π_i.

Efficiency-wise, we generate two new keys for each encryption and only one for additions and revocations. Maintaining several Signal channels requires a number of key generations that grows linearly with the number of devices.

SetUp(1^λ, i)

$sk, pk \leftarrow \mathsf{KeyGen_a}(1^\lambda)$
$\pi_i.sk \leftarrow sk, \pi_i.pk \leftarrow pk$
$\pi_i.1^\lambda \leftarrow 1^\lambda$
return π_i

$Init(\pi_i, s)$

$K_\mathtt{m} \leftarrow \mathsf{KeyGen_m}(\pi_i.1^\lambda)$
$\pi_i^s.K_\mathtt{m} \leftarrow K_\mathtt{m}$
$\pi_i^s.esk \leftarrow \pi_i.sk, \pi_i^s.epk \leftarrow \pi_i.pk$
$\pi_i^s.PK \leftarrow \{\pi_i.pk\}$
$\pi_i^s.devices \leftarrow \{\pi_i.pk\}$
return π_i^s

Enc(m, π_i^s)

$K'_\mathtt{m} \leftarrow \mathsf{KeyGen_m}(\pi_i.1^\lambda)$
Find ind such that $\pi_i^s.PK[ind] = \pi_i^s.epk$
$sk, pk \leftarrow \mathsf{KeyGen_a}(\pi_i.1^\lambda)$
$\pi_i^s.esk \leftarrow sk, \pi_i^s.epk \leftarrow pk$
$\pi_i^s.PK[ind] \leftarrow \pi_i^s.epk$
$\tau \leftarrow \mathsf{Mac}(\text{“}up\text{”} \| \pi_i^s.PK \| K'_\mathtt{m}, \pi_i^s.K_\mathtt{m})$
$c \leftarrow \mathsf{mEnc}(m \| K'_\mathtt{m} \| \tau, \pi_i^s.PK)$
$C_{enc} \leftarrow c \| \pi_i^s.PK, \pi_i^s.K_\mathtt{m} \leftarrow K'_\mathtt{m}$
return C_{enc}, π_i^s

Dec(C_{enc}, π_j^r)

$c \| PK' \leftarrow C_{enc}$
$m \| K_\mathtt{m} \| \tau \leftarrow \mathsf{Dec}(c, \pi_j^r.esk)$
$\mathsf{Verif}(\text{“}up\text{”} \| PK' \| K_\mathtt{m}, \tau, \pi_j^r.K_\mathtt{m})$
$\pi_j^r.K_\mathtt{m} \leftarrow K_\mathtt{m}, \pi_j^r.PK \leftarrow PK'$
return m, π_j^r

Revoke(pk, π_i^s)

Require $pk \neq \pi_i.pk$
Find j such that $\pi_i^s.devices[j] = pk$
if $j = \perp$ **return** $\perp, \pi_i^s$
$D \leftarrow \pi_i^s.devices \setminus \{pk\}$
$\pi_i^s.PK \leftarrow \pi_i^s.PK \setminus \pi_i^s.PK[j]$
$K'_\mathtt{m} \leftarrow\!\!\$\ \mathsf{KeyGen_m}(\pi_i.1^\lambda)$
$\tau \leftarrow \mathsf{Mac}(\text{“}rev\text{”} \| \pi_i^s.PK \| D \| K'_\mathtt{m}, \pi_i^s.K_\mathtt{m})$
$c \leftarrow \mathsf{mEnc}(K'_\mathtt{m} \| \tau, \pi_i^s.PK)$
$C_{rev} \leftarrow c \| \pi_i^s.PK \| D$
$\pi_i^s.K_\mathtt{m} \leftarrow K'_\mathtt{m}, \pi_i^s.devices \leftarrow D$
return C_{rev}, π_i^s

DecRevoke(C_{rev}, π_k^o)

$c \| PK \| D \leftarrow C_{rev}$
$K_\mathtt{m} \| \tau \leftarrow \mathsf{Dec}(c, \pi_k^o.esk)$
$\mathsf{Verif}(\text{“}rev\text{”} \| PK \| D \| K_\mathtt{m}, \tau, \pi_k^o.K_\mathtt{m})$
$\pi_k^o.K_\mathtt{m} \leftarrow K_\mathtt{m}$
$\pi_k^o.PK \leftarrow PK, \pi_k^o.devices \leftarrow D$
return π_k^o

Add&Join($\{pk_{j_\ell}\}_{\ell \in [1,z]}, \pi_i^s$)

Require $(\forall\ \ell \neq \ell', pk_{j_\ell} \neq pk_{j_{\ell'}})$
$\quad \wedge\ \{pk_{j_\ell}\}_{\ell \in [1,z]} \cap \pi_i^s.devices = \emptyset$
$K'_\mathtt{m} \leftarrow \mathsf{KeyGen_m}(\pi_i.1^\lambda)$
$PK' \leftarrow \pi_i^s.PK \cup \{pk_{j_\ell}\}_{\ell \in [1,z]}$
$\pi_i^s.devices \leftarrow \pi_i^s.devices \cup \{pk_{j_\ell}\}_{\ell \in [1,z]}$
$C_{join} \leftarrow$ Join($\{pk_{j_\ell}\}_{\ell \in [1,z]}, PK', K'_\mathtt{m}, \pi_i^s$)

$m_{join} \leftarrow PK' \| \pi_i^s.devices \| K'_\mathtt{m}$
$C_{join} \leftarrow \mathsf{mEnc}(m_{join}, \{pk_{j_\ell}\}_{\ell \in [1,z]})$
return C_{join}

$C_{add} \leftarrow$ Add($PK', K'_\mathtt{m}, \pi_i^s$)

$m_{add} \leftarrow \text{“}add\text{”} \| PK' \| \pi_i^s.devices \| K'_\mathtt{m}$
$\tau \leftarrow \mathsf{Mac}(, \pi_i^s.K_\mathtt{m})$
$\tilde{c} \leftarrow \mathsf{mEnc}(K'_\mathtt{m} \| \tau, \pi_i^s.PK)$
$C_{add} \leftarrow \tilde{c} \| PK' \| \pi_i^s.devices$
return $C_{add},$

$\pi_i^s.PK \leftarrow PK', \pi_i^s.K_\mathtt{m} \leftarrow K'_\mathtt{m}$
return $C_{join}, C_{add}, \pi_i^s$

DecJoin(C_{join}, π_j, r)

$PK \| D \| K_\mathtt{m} \leftarrow \mathsf{Dec}(C_{join}, \pi_j.sk)$
$\pi_j^r.esk \leftarrow \pi_j.sk, \pi_j^r.epk \leftarrow \pi_j.pk, \pi_j^r.K_\mathtt{m} \leftarrow K_\mathtt{m}$
$\pi_j^r.PK \leftarrow PK, \pi_j^r.devices \leftarrow D$
return π_j^r

DecAdd(C_{add}, π_k^o)

$c \| PK \| D \leftarrow C_{join}$
$K_\mathtt{m} \| \tau \leftarrow \mathsf{Dec}(c, \pi_k^o.esk)$
$\mathsf{Verif}(\text{“}add\text{”} \| PK \| D \| K_\mathtt{m}, \tau, \pi_k^o.K_\mathtt{m})$
$\pi_k^o.PK \leftarrow PK, \pi_k^o.devices \leftarrow D, \pi_k^o.K_\mathtt{m} \leftarrow K_\mathtt{m}$
return π_k^o

Figure 4.8 – A Ratcheted Dynamic Multicast construction.

The following theorem enunciates the security of our construction relatively to the RDM security

model described in subsection 4.3.2.

Theorem 4.1. *If* Enc *is an* IND-CCA *secure asymmetric encryption scheme, and* Mac *is secure under multi-instance strong unforgeability, the above construction is a secure ratcheted dynamic multicast for n_d devices, such that, for any PPT adversary making at most q queries to the oracles:*

$$\begin{aligned}\mathsf{Adv}^{\text{RDM-IND}}_{\mathsf{RDM},n_d,q,\mathcal{A}}(\lambda) &= \left|\Pr\left[Exp^{\text{RDM-IND}}_{\mathsf{RDM},n_d,q,\mathcal{A}}(n)\right] - \frac{1}{2}\right| \\ &\leq q \cdot \text{mi-SUF} + (q+1) \cdot n_d \cdot \text{IND-CCA}.\end{aligned}$$

In practice, one would use hybrid encryption (as described in section 2.3.1) instead of a single asymmetric encryption scheme in this construction. It means that the asymmetric encryption is used to transmit a common symmetric key to all devices and that data are then encrypted with this key. This would modify the security argument only by a negligible term due to the symmetric encryption. We decide to present our construction with the asymmetric part so as not to add extra lines in an already complex construction.

A game based proof We now give a proof of Theorem 4.1.

Proof. The proof of correctness follows by inspection: if the messages are correctly transmitted, then the devices all obtain the correct updated keys to verify the Mac and decrypt the messages.

For the indistinguishably, we consider a proof by iteration. Considering a session (i, s), we call a step fresh if it is suitable for challenge. We show that considering any session, a fresh step will always correspond to a group of honest devices with secret keys unknown to the adversary and a safe shared MAC key. As for the Signal proof, we obtain a tree of possible sequences of oracle calls. Hence for each step, we consider the different branches of the tree that the adversary can follow.

- If the considered step is fresh with a secret MAC key, then $\mathcal{A}$ cannot interfere in the protocol (thanks to SUF of the MAC apart from the initial step where specific restrictions exist) and the step after any action of $\mathcal{A}$ is either fresh with a secure MAC key (thanks to the IND-CCA of the encryption) or non-fresh.
- If the considered step is not fresh, then $\mathcal{A}$ cannot interfere due to restrictions imposed by the model. The step after any action of $\mathcal{A}$ is either a non fresh step, or a fresh step with a secret MAC key if the corruption has been healed.

From this, we obtain that in any session, any step suitable for challenge is necessarily uncompromised.

In more detail, we consider a lookup table $\mathbb{B}$ in which we will write each RevealLongTerm or RevealState query made by the adversary. RevealLongTerm and RevealState are written and erased in $\mathbb{B}$ the same way corresponding flags are turned to $false$ or $true$, but we need this table to identify the last reveal/corruption healing. We choose to observe the experiment from one session (i, s) but what we actually look at is the behaviour of the group at each step. Hence we consider in the lookup table lines that concerns (i, s) or any matching session. If $\mathbb{B}$ contains only lines for non matching session, we say it is empty. Of course adversary does not only play with (i, s) but also with others. We therefore consider queries on (i, s) or on other sessions (j, r) if it is a matching sessions because it has an effect on (i, s) state. In the proof, we denote by S_i the event "the adversary wins in the game i".

Case of initial sessions

Initialization. We consider an initial session (i, s). We initialize our iteration by considering step 0 of the session. As (i, s) is an initial session, its step 0 concerns only device i. First, a MAC key K_m is generated and is not used at this moment, so it can be considered as non revealed.

Case A: fresh departure. Step 0 is not corrupted nor revealed. $\mathbb{B}$ is empty. It means i is not corrupted and K_m is safe. As K_m is safe, we show $\mathcal{A}$ can not interfere.
Game A.0. Let Game 0 be the original game with this configuration.
Game A.1. Let Game A.1 be as Game 0 except the simulator respectively reject all ciphertexts C_{enc} and C_{add} not produced by OEnc and OAdd&Join on input to ODec and ODecAdd (Revocation is not considered here as i cannot revoke himself). Otherwise, we can use it to build an adversary against strong unforgeability of the MAC scheme. We obtain:

$$|\Pr[S_{A.1}] - \Pr[S_{A.0}]| \leq \text{mi-SUF}.$$

At this point, we consider there are no forgeries, this means that all ciphertexts accepted[6] by different oracles are produced by other oracles and $\mathcal{A}$ cannot join or send encrypted message maliciously.

From there, $\mathcal{A}$ can query for an OEnc, OAdd&Join, ORevoke, RevealState or RevealLongTerm. We do not consider decryptions here as there is still no ciphertext produced by oracles OEnc, OAdd&Join, and ORevoke. We study each option:

- RevealLongTerm(i). This query will lead to a non fresh step (step has not changed, it just got corrupted) and (i, s) is written as corrupted in $\mathbb{B}$ (actually, we shall write (i, s') for all sessions runned by i, but, since we are only interested in (i, s) now, we only consider this one).

- RevealState$(i, s, 0)$. This query will lead to a non fresh step and (i, s) is written as corrupted in $\mathbb{B}$.

- Enc(m, i, s). Let **Game A.2.Enc** be as Game A.1 but ensures $\mathcal{A}$ learns nothing about the newly generated MAC key K'_m. That is we replace the ciphertext $c = \text{EncAsym}(m\|K'_m\|\tau, \pi_i^s.PK)$ produced in OEnc by a random $c' = \text{EncAsym}(rand, \pi_i^s.PK)$ and keep the couple (ciphertext, plaintext) = $(c', m\|K'_m\|\tau)$ in a list L for later decryption. i is not corrupted and is the only receiver of this encryption ($\pi_i^s.PK = \pi_i.pk$). IND-CCA security[7] of the encryption scheme ensures we can do this substitution properly with loss:

$$|\Pr[S_{A.2.Enc}] - \Pr[S_{A.1}]| \leq \text{IND-CCA}.$$

  We are in a fresh state with secure MAC key K_m.

[6] For ciphertexts accepted by the ODecJoin, the security model already restricts the ciphertext to belong to P (line 1. of this oracle). This is due to the fact that we do not have made a choice to authenticate a device to another. By considering a specific solution to authenticate them, we could delete this restriction in the security model and describe here an additional game A.1' in which we would apply the security of this chosen authentication to obtain C_{join} on input to ODecJoin has been produced by OAdd&Join.

[7] This actually corresponds to a ror-CCA experiment. The reduction from ror-CCA to IND-CCA given in subsection 2.3.1 enables us to keep the more classical notion of IND-CCA.

- OAdd&Join $\left(\{j_\ell\}_{\ell\in[1,z]}, i, s\right)$ with none of the j_ℓ corrupted. Consider a **Game A.2.Join** that runs as Game A.1 except that we ensure that $\mathcal{A}$ learns nothing about the newly generated MAC key K'_m. We replace the ciphertext $C_{join} = \mathsf{EncAsym}(PK'\|K'_m, \{pk_{j_\ell}\}_{\ell\in[1,z]})$ produced in Join of OAdd&Join by $C_{join} = \mathsf{EncAsym}(rand, \{pk_{j_\ell}\}_{\ell\in[1,z]})$ and ciphertext $\tilde{c} = \mathsf{EncAsym}(K'_m\|\tau, \pi_i^s.PK)$ produced in Add of OAdd&Join by $\tilde{c}' = \mathsf{EncAsym}(rand, \pi_i^s.PK)$. We keep both couples $(C'_{join}, PK'\|K'_m, \{pk_{j_\ell}\}_{\ell\in[1,z]})$ and $(\tilde{c}', K'_m\|\tau, \pi_i^s.PK)$ in the list L for later decryption. None of the $\{j_\ell\}_{\ell\in[1,z]}$ is corrupted and they are the only receivers of the encryption C_{join} and i is not corrupted and is the only receiver of the encryption $\tilde{c}$. IND-CCA security of the encryption scheme - extended to parallel encryption - ensures we can do both substitutions properly with loss:

$$\begin{aligned}|\Pr[S_{A.2.Join}] - \Pr[S_{A.1}]| &\leq \text{IND-CCA} + z \cdot \text{IND-CCA} \\ &\leq n_d \cdot \text{IND-CCA}.\end{aligned}$$

  We are in a fresh state with secure MAC key K_m.

- OAdd&Join$(\{j_\ell\}_{\ell\in[1,z]}, i, s)$ with one or more j_ℓ corrupted. For each device j_ℓ corrupted, $(j_\ell, allsessions)$ is written as corrupted in $\mathbb{B}$. Adversary obtains the new MAC key K'_m through the joining process and we are in a non fresh step with unsafe MAC key.

Case B: unfresh departure. If i is corrupted, (i, s) is written as corrupted in $\mathbb{B}$ and step 0 is not suitable for a Challenge query.
As before, $\mathcal{A}$ can query for an OEnc, OAdd&Join, and RevealState (a query RevealLongTerm is useless here as i is already corrupted). We do not consider decryption oracles here as there is still no ciphertext produced by oracles OEnc, OAdd&Join, and ORevoke and $\mathcal{A}$ cannot join, revoke, or send encrypted message maliciously. This last point is due to the natural restrictions in the security model: the model does not allow $\mathcal{A}$ to forge a message after a corruption or an exposure.
Game B.0. Let Game 0 be the original game with this configuration. We study each option:

- RevealState$(i, s, 0)$. This step remains a non fresh step and i is still written as corrupted in $\mathbb{B}$.

- OEnc(m, i, s). Let **Game B.1.Enc** be as Game 0 but ensures $\mathcal{A}$ learns nothing about the newly generated MAC key K'_m. We replace the ciphertext $c = \mathsf{EncAsym}(m\|K'_m\|\tau, \pi_i^s.PK)$ produced in OEnc by $c' = \mathsf{EncAsym}(rand, \pi_i^s.PK)$ and keep the couple $(c', m\|K'_m\|\tau)$ in the list L for later decryption. i is corrupted, but during the execution of OEnc, this oracle generates for i a new secret/public key (sk, pk) (line 3. of Enc) unknown to $\mathcal{A}$ and we can delete the entry (i, s) from the table $\mathbb{B}$. $\mathbb{B}$ is now empty. The ciphertext c is intended for the sole new key of i, non corrupted at this step. IND-CCA security of the encryption scheme ensures we can do this substitution properly with loss:

$$|\Pr[S_{B.1.Enc}] - \Pr[S_{B.0}]| \leq \text{IND-CCA}.$$

  We are in a fresh state with secure MAC key K_m.

- OAdd&Join$(\{j_\ell\}_{\ell\in[1,z]}, i, s)$ with none of the j_ℓ corrupted. As $\mathcal{A}$ get the new MAC key K'_m through the Add part of OAdd&Join, we demand no security on the private side apart from the model restrictions. We are in a non fresh state with unsafe MAC key.

- OAdd&Join$(\{j_\ell\}_{\ell\in[1,z]}, i, s)$ with one or more j_ℓ corrupted. For each j_ℓ corrupted, (j_ℓ,$allsessions$) is written as corrupted in $\mathbb{B}$. Adversary obtains the new MAC key K'_m through the joining process and we are in a non fresh step with unsafe MAC key.

Iteration. We now consider a session (i, s) at step t in the protocol after q queries. Each reveal or corrupt has been registered in $\mathbb{B}$. As before, we consider case A when we start from a fresh step and case B when we start from a non fresh step.

Case A: fresh departure ($\mathbb{B}$ is empty).
Game A.0. Let Game 0 be the original game with this configuration.
Game A.1. Let Game A.1 be as Game 0 except the simulator respectively reject all ciphertexts C_{enc}, C_{add}, and C_{rev} not produced by OEnc, OAdd&Join, and ORevoke on input to ODec, ODecAdd, and ORevoke. Otherwise, we can use it to build an adversary against strong unforgeability of the MAC scheme. We obtain:

$$|\Pr[S_{A.1}] - \Pr[S_{A.0}]| \leq \text{mi-SUF}.$$

At this point, all ciphertexts accepted[8] by different oracles are produced by other oracles and $\mathcal{A}$ cannot join or send encrypted message maliciously.

From there $\mathcal{A}$ can query for an OEnc, OAdd&Join, ORevoke, RevealState, or RevealLongTerm. Decryption oracles do not induce changes on the honesty of the devices, on the secret of the MAC key, or on the freshness of the step due to Game A.1, so we do not detail them. We note $t = \pi_i^s.\texttt{step}$ We study each option:

- RevealLongTerm(i). If i is active in the session - meaning that it already does not use his long term secret key $\pi_i.sk$ to communicate with the group of devices -, then nothing happens. Else (i, s) is written as corrupted in $\mathbb{B}$.
- RevealLongTerm(j). If there exists r, $t_s \leq t$ such that $(j, r, \pi_j^r.\texttt{step})$ is non active and matches (i, s, t_s), (j, r) is written as corrupted in $\mathbb{B}$. Otherwise, as j is active on all sessions matching with (i, s), nothing happens.
- RevealState(i, s). (i, s) is written as corrupted in $\mathbb{B}$.
- RevealState(j, r). If there exists $t_s \leq t$ such that $(j, r, \pi_j^r.\texttt{step})$ matches (i, s, t_s), (j, r) is written as corrupted in $\mathbb{B}$. If $t_s < t$, then the MAC key has changed but as the secret key of j is known to the adversary, he has access to the new MAC key also.
- OEnc(m, i, s). Let **Game A.2.Enc** be the same as Game A.1 but ensures $\mathcal{A}$ learns nothing about the newly generated MAC key K'_m. That is we replace the ciphertext produced in OEnc, $c = \text{EncAsym}(m\|K'_m\|\tau, \pi_i^s.PK)$, by $c' = \text{EncAsym}(rand, \pi_i^s.PK)$ and keep the couple $(c', m\|K'_m\|\tau)$ in the list L for later decryption. $\mathbb{B}$ is empty, IND-CCA security of the encryption scheme ensures we can do this substitution properly with loss $n_{|PK|}$ where $n_{|PK|}$ is the number of public keys in $\pi_i^s.PK$, equals to the number of devices in the group at this step:

$$\begin{aligned}|\Pr[S_{A.2.Enc}] - \Pr[S_{A.1}]| &\leq n_{|PK|} \cdot \text{IND-CCA} \\ &\leq n_d \cdot \text{IND-CCA}.\end{aligned}$$

We are in a fresh state with secure MAC key K_m.

[8] For ciphertexts accepted by the ODecJoin, as already explained, $\mathcal{A}$ cannot join a maliciously $DecJoin$ due to the security model.

- $\mathsf{OEnc}(m, j, r)$. If (j,r,$\pi_j^r.step$) matches $(i, s, t_s < t)$ then the message produced is not valid. In fact, (j, r) matches on an older MAC key than the one (i, s) uses now. If (j,r,$\pi_j^r.step$) matches (i, s, t), just process the same as for $\mathsf{OEnc}(m, i, s)$. Otherwise do nothing.

- $\mathsf{OAdd\&Join}(\{k_\ell\}_{\ell\in[1,z']}, i, s)$ with none of the k_ℓ corrupted. Let the next **Game A.2.Join** be as Game A.1 but ensures $\mathcal{A}$ learns nothing about the newly generated MAC key K'_m. That is we replace the ciphertext $C_{join} = \mathsf{EncAsym}(PK'\|K'_m, pk_k)$ produced in the Join subprocedure of $\mathsf{OAdd\&Join}$ by $C_{join} = \mathsf{EncAsym}(rand, pk_k)$ and the cipher $\tilde{c} = \mathsf{EncAsym}(K'_m\|\tau, \pi_i^s.PK)$ produced in Add of $\mathsf{OAdd\&Join}$ by $\tilde{c}' = \mathsf{EncAsym}(rand, \pi_i^s.PK)$. We keep both couples $(C'_{join}, PK'\|K'_m, pk_k)$ and $(\tilde{c}', K'_m\|\tau, \pi_i^s.PK)$ in the list L for later decryption. None of the k_ℓ is corrupted and they are the only receivers of the encryption C_{join}. $\mathbb{B}$ is empty which means that the receivers of $\tilde{c}$ are not corrupted. IND-CCA security of the encryption scheme ensures we can do both substitutions properly with loss where $n_{|PK|}$ is the number of public keys in $\pi_i^s.PK$, equals to the number of devices in the group:

$$\begin{aligned} |\Pr[S_{A.2.Join}] - \Pr[S_{A.1}]| &\leq (n_{|PK|} + z') \cdot \text{IND-CCA} \\ &\leq n_d \cdot \text{IND-CCA}. \end{aligned}$$

  We are in a fresh state with secure MAC key K_m.

- $\mathsf{OAdd\&Join}(\{k_\ell\}_{\ell\in[1,z']}, i, s)$ with one or more k_ℓ corrupted. For each device k_ℓ corrupted, $(k_\ell, allsessions)$ is written as corrupted in $\mathbb{B}$. Adversary obtains the new MAC key K'_m through the joining process and we are in a non fresh step with unsafe MAC key.

- $\mathsf{OAdd\&Join}(\{k_\ell\}_{\ell\in[1,z']}, j, r)$. If ($j$,$r$,$\pi_j^r.step$) matches $(i, s, t_s < t)$ then the message produced is not valid. In fact, (j, r) matches on an older MAC key than the one (i, s) uses now. If (j,r,$\pi_j^r.step$) matches a session (i, s, t), just processes the same as for the previous case $\mathsf{OAdd\&Join}(\{k_\ell\}_{\ell\in[1,z']}, i, s)$. Otherwise do nothing.

- $\mathsf{ORevoke}(pk, i, s)$. The construction verifies that $\exists k$ such that $pk = pk_k$[9]. $\mathbb{B}$ is empty, we revoke k even if it was not written as corrupted - in reality this allows to prevent a further corruption. Let **Game A.2.Rev** be as Game A.1 but ensures $\mathcal{A}$ learns nothing about the newly generated MAC key K'_m. Again, we replace the ciphertext $c = \mathsf{EncAsym}(K'_m\|\tau, \pi_i^s.PK)$ produced in $\mathsf{ORevoke}$ by $c' = \mathsf{EncAsym}(rand, \pi_i^s.PK)$ and keep the couple $(c', K'_m\|\tau)$ in the list L for later decryption. $\mathbb{B}$ is empty, IND-CCA security of the encryption scheme ensures we can do this substitution properly with loss $n_{|PK|}$ where $n_{|PK|}$ is the number of public keys in $\pi_i^s.PK$, equals to the number of devices in the group:

$$\begin{aligned} |\Pr[S_{A.2.Rev}] - \Pr[S_{A.1}]| &\leq n_{|PK|} \cdot \text{IND-CCA} \\ &\leq n_d \cdot \text{IND-CCA}. \end{aligned}$$

  We are in a fresh state with secure MAC key K_m.

- $\mathsf{ORevoke}(pk_k, j, r)$. If ($j$,$r$,$\pi_j^r.step$) matches $(i, s, t_s < t)$ then the message produced is not valid. If (j,r,$\pi_j^r.step$) matches (i, s, t),just processes the same as $\mathsf{ORevoke}(pk_k, i, s)$. Otherwise do nothing.

[9] In the following, we assume that revoke queries are always on a valid long term key pk_k.

Case B: unfresh departure. If $\mathbb{B}$ is not empty, there is at least an identifier (device, session) written as corrupted in $\mathbb{B}$ and step t is not suitable for the Challenge query.
As before, $\mathcal{A}$ can query for an OEnc, OAdd&Join, ORevoke, RevealState, and RevealLongTerm. As for the case B of the initialization, $\mathcal{A}$ cannot join, revoke, or send encrypted message maliciously. Again, this is due to natural restrictions in the security model.
Game B.0. Let Game 0 be the original game with this configuration. We study each option:

- RevealLongTerm(i). If i is active in the session - meaning that it already does not use his long term secret key $\pi.sk$ to communicate with the group of devices -, then nothing happens. Else (i, s) is written as corrupted in $\mathbb{B}$.
- RevealLongTerm(j). If there exists r, $t_s \leq t$ such that $(j, r, \pi_j^r.\texttt{step})$ is non active and matches (i, s, t_s), (j, r) is written as corrupted in $\mathbb{B}$. Otherwise, as j is active on all sessions matching with (i, s), nothing happens.
- RevealState(i, s). (i, s) is written as corrupted in $\mathbb{B}$.
- RevealState(j, r). If there exists $t_s \leq t$ such that $(j, r, \pi_j^r.step)$ matches (i, s, t_s), (j, r) is written as corrupted in $\mathbb{B}$. If $t_s < t$, then the MAC key has changed but as the secret key of j is know to adversary, he will access to the new MAC key also.
- OEnc(m, i, s). If (i, s) is the sole entry written as corrupted in $\mathbb{B}$, then as for the case B in step 0, we regain security. (i, s) is now not written as corrupted in $\mathbb{B}$. Let in this case, **Game B.1.Enc** be as Game 0 but ensures $\mathcal{A}$ learns nothing about the newly generated MAC key K'_m. We replace the ciphertext $c = \mathsf{EncAsym}(m\|K'_m\|\tau, \pi_i^s.PK)$ produced in OEnc by $c' = \mathsf{EncAsym}(rand, \pi_i^s.PK)$ and keep the couple $(c', m\|K'_m\|\tau)$ in the list L for later decryption. $\mathbb{B}$ is empty, IND-CCA-security of the encryption scheme ensures we can do this substitution properly with loss $n_{|PK|}$ where $n_{|PK|}$ is the number of public key in $\pi_i^s.PK$, equals to the number of devices in the group:

$$\begin{aligned} |\Pr[S_{B.1.Enc}] - \Pr[S_{B.0}]| &\leq n_{|PK|} \cdot \text{IND-CCA} \\ &\leq n_d \cdot \text{IND-CCA}. \end{aligned}$$

  We are in a fresh state with secure MAC key K_m.
  If $\mathbb{B}$ contains lines other than (i, s), security is still not back. (i, s) is erased from $\mathbb{B}$ if it was previously written. Nothing more is expected since the new MAC key K'_m is accessible through remained corrupted/revealed devices and we are in a non fresh state.
- OEnc(m, j, r). If (j,r,$\pi_j^r.step$) matches ($i, s, t_s < t$) then the message produced is not valid, do nothing. If (j,r,$\pi_j^r.step$) matches (i, s, t), just process the same as for OEnc(m, i, s). Otherwise do nothing.
- OAdd&Join$(\{k_\ell\}_{\ell\in[1,z']}, i, s)$ with none of the $\{k_\ell\}_{\ell\in[1,z']}$ corrupted. We do nothing as the adversary will obtain the new MAC key K'_m through the ciphertext C_{add}. We remain in a non fresh step with unsafe MAC key.
- OAdd&Join$(\{k_\ell\}_{\ell\in[1,z']}, i, s)$ with one or more of the $\{k_\ell\}_{\ell\in[1,z']}$ corrupted. For all corrupted k_ℓ we write the newly generated (k_ℓ, r_ℓ) as corrupted in $\mathbb{B}$ as soon as DecJoin(k_ℓ, c_{join}) is queried on the returned C_{join} message and do nothing else as the adversary is able to obtain the new MAC key K'_m. We remain in a non fresh step with unsafe MAC key.

- $\mathsf{OAdd\&Join}(\{k_\ell\}_{\ell\in[1,z']}, j, r)$. If $(j,r,\pi_j^r.step)$ matches $(i,s,t_s < t)$ then the message produced is not valid, do nothing. If $(j,r,\pi_j^r.step)$ matches (i,s,t), just process the same as for the previous case $\mathsf{OAdd\&Join}(\{k_\ell\}_{\ell\in[1,z']}, i, s)$. Otherwise do nothing.

- $\mathsf{ORevoke}(pk_k, i, s)$. For all o, $t_o, t_s \leq t$ such that (k,o) is written as corrupted in $\mathbb{B}$ and (k,o,t_o) is matching (i,s,t_s), we erase (k,o) from $\mathbb{B}$. If (k,o) was the sole entry written as corrupted in $\mathbb{B}$, then as for the case B in step 0, we regain security. Let in this case, **Game B.1.Rev** be as Game 0 but ensures $\mathcal{A}$ learns nothing about the newly generated MAC key K'_m. We replace the ciphertext $c = \mathsf{EncAsym}(K'_m\|\tau, \pi_i^s.PK)$ produced in $\mathsf{ORevoke}$ by $c' = \mathsf{EncAsym}(rand, \pi_i^s.PK)$ and keep the couple $(c', K'_m\|\tau)$ in the list L for later decryption. $\mathbb{B}$ is now empty, IND-CCA security of the encryption scheme ensures we can do this substitution properly with loss $n_{|PK|}$ where $n_{|PK|}$ is the number of public keys in $\pi_i^s.PK$, equals to the number of devices in the group:

$$|\Pr[S_{B.1.Rev}] - \Pr[S_{B.0}]| \leq n_d \cdot \text{IND-CCA}.$$

  We are in a fresh state with secure MAC key K_m. If $\mathbb{B}$ contains other lines not concerning (k,o), security is still not back. Nothing more is attended since the new MAC key K'_m is accessible through remained corrupted/revealed devices and we are in a non fresh state.

- $\mathsf{ORevoke}(pk_k, j, r)$. If $(j,r,\pi_j^r.step)$ matches $(i,s,t_s < t)$ then do nothing. If $(j,r,\pi_j^r.step)$ matches (i,s,t), just process the same as for $\mathsf{ORevoke}(pk_k, i, s)$. Otherwise do nothing.

Case of non initial sessions

Through restrictions, a session can only be created by the OInit. This means that every session is chained with an initial session. The only modification in our game hops is that we now consider chained session instead of solely matching sessions. (For instance in a case $\mathsf{OEnc}(m, j, r)$, replace "If $(j,r,\pi_j^r.step)$ matches $(i,s,t_s < t)$ then the message produced is not valid, do nothing" by "If $(j,r,\pi_j^r.step)$ is chained with (i,s,t) then the message produced is not valid, do nothing.") The above game hops are still valid because one can go from the initial session to the one of our interest following a chain of sessions. As we consider actions on chained sessions and as those chains are taken into account in our restrictions, the same logic applies.

Consider the event E: after q queries any session state is either corrupted or a fresh state with no adversary having interfered. Suppose, n_d is a fixed value that corresponds to the maximum number of devices the adversary can play with, we obtain:

$$\Pr[E] \leq q \cdot \text{mi-SUF} + q \cdot n_d \cdot \text{IND-CCA}.$$

Finally when adv queries a $\mathsf{Challenge}$, if the step is not fresh, challenge is not valid. If the step is fresh with secret keys and no adversary having infiltrated the group, IND-CCA security of the encryption scheme used n_d parallel times gives:

$$\begin{aligned}\mathsf{Adv}^{\text{RDM-IND}}_{\mathsf{RDM},n_d,q,\mathcal{A}}(\lambda) &= \left|\Pr\left[Exp^{\text{RDM-IND}}_{\mathsf{RDM},n_d,q,\mathcal{A}}(n)\right] - \frac{1}{2}\right| \\ &\leq \Pr[E] + n_d \cdot \text{IND-CCA} \\ &\leq q \cdot \text{mi-SUF} + (q+1) \cdot n_d \cdot \text{IND-CCA}.\end{aligned}$$

□

4.4 A Multi-Device Messaging protocol

In this section, we focus on our multi-device version of the Signal protocol and on its security analysis. We start with a description of the protocol, which we define as a Multi-Device Instant Messaging protocol (MDIM), and the associated security model, before giving a construction and a proof of the security of our protocol relatively to the model. The security model we consider is a direct combination of [CCD+17] and of our RDM model described in subsection 4.3.2. As the model proposed by Cohn-Gordon *et al.* is tailored for Signal (through its freshness predicate,) so is ours. Since several other models have been proposed for RKE, each with its pros and cons, we do not pretend to present a universal model for multi device RKE. However, as our model construction is modular, we expect it to adapt well to other RKE models.

4.4.1 A formal MDIM

Before going into the model details, we first need to give a high level description of the protocol concerned by the model. A MDIM is a state full protocol between two users, an initiator $\mathcal{P}_i$ and a responder $\mathcal{P}_r$ that both possess many devices $d_{in,1}, \ldots, d_{in,n_{in}}$ (respectively $d_{r,1}, \ldots, d_{r,n_r}$). Each participant possesses a general key package composed of a long-term user key and possibly ephemeral keys. This user key package is identified as the user's prekey bundle $(sprekey, pprekey)$. Each device has its own device long term key dsk. Each device $\mathtt{d}_{u,i}$ maintains a state $\pi_{u,i}$ and each session s executed by a device $\mathtt{d}_{u,i}$ maintains a session state $\pi^s_{u,i}$. The device state aggregates all non session-specific elements:

- dID, the device identifier.
- uID, the user identifier.
- dsk, the device's secret key.
- $sprekeys$, the user's secret keys. This comprises user long term key usk and the ephemeral keying material $ephsk$ needed for initialization (correspondingly, $pprekey$ contains both upk and $ephpk$).
- $Devices$, the public keys of the owner other devices.
- $Sessions$, a list of all sessions the device i is engaged in.

A session state $\pi^s_{u,i}$ gathers session specific information. The session state has two facets, one that registers the data needed for the conversation with the peer:

- $role$, the role of the user u: initiator or receiver,
- $peer$, the intended peer user of this session,
- $rand$, the current ratchet secret,
- $rand_{peer}$, the current public ratchet value of the intended peer,
- $sessionkey$, the current messaging session key,
- $state$, all other secret information needed,

and another that maintains the data shared with other devices:

- $devrand$, the randomness,
- PK a public key shared with the other devices.

The choice of this description is mainly due to our will to obtain a protocol that can be built as the composition of two protocols, one between the peers, and another between the devices. However, one can imagine a protocol where only one randomness is needed for instance. A session state $\pi^s_{u,i}$ has access to general information of the device state $\pi_{u,i}$. Conversely, a device state $\pi_{u,i}$ gives implicit access to every session state $\pi^s_{u,i}$.

Protocol description. A MDIM is defined as the following tuple of algorithms:

- $\mathsf{UserKeyGen}(1^\lambda) \to pprekeys, sprekeys$. Generates two lists of prekeys. These lists gather respectively public and private user keys, and optional extra keying material. We call these keys the "prekey bundle".
- $\mathsf{DeviceSetUp}(1^\lambda, (u,i)) \to \pi_{u,i}$. Generates the device key and records them in a new state $\pi_{u,i}$.
- $\mathsf{Register}(\mathsf{uID}, \pi_{u,i}) \to \pi_{u,i}$. Creates the prekey bundle and uID, and registers them together with the device key on the server. Completes the device state $\pi_{u,i}$ with the prekey bundle and uID.
- $\mathsf{InitSession}(role, pprekeys_v, \pi_{u,i}, s) \to C_{init}, c_{out}, \pi^s_{u,i}$. On input a $role$, the $pprekeys_v$ of the intended peer v, a device state $\pi_{u,i}$, and a session identifier s, returns a (eventually empty) ciphertext C_{init} for u's other devices, a (possibly empty) ciphertext c_{out} for the intended peer, and a session state $\pi^s_{u,i}$.
- $\mathsf{ReceiveInitSession}(C_{init}, \pi_{u,j}, r) \to \pi^r_{u,j}$. On input a ciphertext C_{init}, a device state $\pi_{u,j}$, and a session identifier r, outputs a session state $\pi^r_{u,j}$.
- $\mathsf{Send}(m, \pi^s_{u,i}) \to C_{in}, c_{out}, \pi^s_{u,i}$. On input a plaintext m and a session state $\pi^s_{u,i}$, returns one ciphertext C_{in} for u's other devices, a second ciphertext c_{out} for the intended peer $\pi^s_{u,i}.peer$, and an updated state $\pi^s_{u,i}$.
- $\mathsf{ReceiveIn}(C_{in}, \pi^r_{u,j}) \to m, \pi^r_{u,j}$. On input a ciphertext C_{in} and a session state $\pi^r_{u,j}$, outputs a (eventually empty) message m and an updated session state $\pi^r_{u,j}$.
- $\mathsf{ReceiveOut}(c_{out}, \pi^p_{v,\ell}) \to m, \pi^p_{v,\ell}$. On input a ciphertext c_{out} and a session state $\pi^p_{v,\ell}$, outputs a (potentially empty) message m and an updated session state $\pi^p_{v,\ell}$.
- $\mathsf{Add\&Join}(dpk_j, \pi_{u,i}) \to \{C_{join,s}, C_{add,s}, c_{out,s}\}_{s\in S}, \pi_{u,i}$
  with $S = \pi_{u,i}.Sessions$. On input a device public key dpk_j and a device state $\pi_{u,i}$, returns, for each session s the device i is engaged in, one ciphertext $C_{join,s}$ for the new device j, one ciphertext $C_{add,s}$ for all the other devices, a ciphertext $c_{out,s}$ for the intended peer, and an updated state $\pi_{u,i}$ (comprising updated session states $\pi^s_{u,i}$ for each s).
- $\mathsf{DecJoin}\left(\{C_{join,r}\}_{r\in[1,R]}, \pi_{u,j}\right) \to \pi_{u,j}$. On input R ciphertexts $C_{join,r}$ and a device state $\pi_{u,j}$, returns an updated device state $\pi_{u,j}$ (comprising R new session states $\pi^r_{u,j}$).
- $\mathsf{DecAdd}\left(\{C_{add,o}\}_{o\in[1,O]}, \pi_{u,k}\right) \to \pi_{u,k}$. On input O ciphertexts $C_{add,o}$ and a device state $\pi_{u,k}$, returns an updated device state $\pi_{u,k}$ (comprising updated session states $\pi^o_{u,k}$ for each o).

- $\mathsf{Revoke}(dpk, \pi_{u,i}) \rightarrow \{C_{rev,s}, c_{out,s}\}_{s \in \pi_{u,i}.Sessions}, pprekeys, \pi_{u,i}$. On input a device public key dpk and a user state $\pi_{u,i}$, returns, for each session s the device i is engaged in, a ciphertext $c_{rev,s}$ for all the other devices of user u and a ciphertext $c_{out,s}$ for the intended peer, and a new user state $\pi_{u,i}$ (comprising updated session states $\pi^s_{u,i}$ for each s).
- $\mathsf{DecRevoke}\left(\{C_{rev,o}\}_{o \in [1,O]}, \pi_{u,k}\right) \rightarrow \pi_{u,k}$. On input O ciphertexts $C_{rev,o}$ and a device state $\pi_{u,k}$, returns an updated device state $\pi_{u,k}$ (comprising updated sessions states $\pi^o_{u,k}$ for each o).

Once again, this description is not as high-level as it could be. All the receiving algorithms could be gathered in a general Receive algorithm for instance, in order to obtain a less construction-oriented description of the protocol. We adopt the detailed position on purpose. Firstly, it facilitates the exposure of the security model, in particular the freshness conditions. Secondly, as explained in the section introduction, we do not aim at providing a universal model, but more a model that considers the composition of a RDM with a RKE. Gathering the appropriate algorithms in a single Receive and writing the freshness conditions aside would give a more general (for a non composition based protocol for instance) but less clear presentation.

4.4.2 A composed security model.

Now that we have a formal description of the protocol, we detail an associated security model. As before, this model shall formalize the security properties that one can expect from any MDIM protocol.

The environment. Our model considers concurrent execution of the protocols. Hence we have a set of users $\{\mathcal{P}_1, \ldots, \mathcal{P}_{n_U}\}$ and for each user $\mathcal{P}_u$, $\{\mathtt{d}_{u,1}, \ldots, \mathtt{d}_{u,n_d}\}$ is the set of his devices. Each device $\mathtt{d}_{u,i}$ is modelled by an oracle $\pi_{u,i}$ and each session s executed by a device $\mathtt{d}_{u,i}$ is modelled by an oracle $\pi^s_{u,i}$. Device oracles maintain device states and session oracles maintain session states as defined in subsection 4.4.1. In the following, we assimilate device or session oracles and their state.

Protocol steps. As in the RDMmodel described in subsection 4.3.2, we consider the steps of the protocol. Each Send, Add, Revoke, or corresponding Receive or Dec algorithm brings the session to a new step and corresponding oracles will increment the step counter.

If no change occurs, values are transmitted from one step to another (e.g. if state does not change at step t_s, then $\pi^s_{u,i}[t_s + 1].state = \pi^s_{u,i}[t_s].state$). We refer to session s run by $\pi_{u,i}$ at step t_s as (u, i, s, t_s).

Partnered and matching sessions. In order to define the matching between sessions run by different users, we first need to consider the relationship between devices belonging to a single user. We introduce the notion of partnered sessions. For this definition, we still consider a matching based on the transcript but we separate the conversation between the devices (written in a session identifier $\mathtt{sid}_1$) from the messages sent and received from a peer (gathered in $\mathtt{sid}_2$). Partnered sessions correspond to devices of a single user that are online at the same moment. We define the session identifier $\mathtt{sid}_1$ as the concatenation of ciphertexts C_{in} sent by $\mathsf{Send}(\cdot, \pi^s_{u,i})$, $\mathsf{Add\&Join}$, or Revoke or received through $\mathsf{ReceiveIn}(\cdot, \pi^{\mathtt{s}}_{\mathtt{u,i}})$, $\mathsf{DecAdd}(\cdot, \pi^s_{u,i})$, $\mathsf{DecJoin}(\cdot, \pi^s_{u,i})$, or $\mathsf{DecRevoke}(\cdot, \pi^s_{u,i})$. We write $\pi^s_{u,i}[t_s].\mathtt{sid}_1 = C^1_{u,i}[0] \| C^1_{u,i}[1] \| \ldots \| C^1_{u,i}[t_s]$. Hence we can write the definition of partnering as follow:

Definition 4.5 (Partnered sessions at some step.)**.** *Two sessions* (u, i, s, t_s) *and* (u, j, r, t_r) *are partnered if:*

- $\pi^s_{u,i}.role = \pi^r_{u,j}.role$,
- $\pi^s_{u,i}.peer = \pi^r_{u,j}.peer$,
- $\pi^s_{u,i}.uID = \pi^r_{u,j}.uID$,
- *there exists* $\mathtt{sid}'$ *subset of* $\pi^s_{u,i}[t_s].\mathtt{sid_1}$ *such that* $\pi^s_{u,i}[t_s].\mathtt{sid_1} \tilde{=} \mathtt{sid}' \| \pi^r_{u,j}[t_r].\mathtt{sid_1}$ *(or reversely for* $\pi^s_{u,i}$ *and* $\pi^r_{u,j}$ *if* j *was there for longer than* i*), where* $\tilde{=}$ *is defined as follow:* $\pi^s_{u,i}[t_s].\mathtt{sid_1} \tilde{=} \mathtt{sid}' \| \pi^r_{u,j}[t_r].\mathtt{sid_1}$ *if,* $\forall t \in [0; t_r]$*:*
  - *either* $C^1_{u,i}[|t_s - t_r| + t] = C^1_{u,j}[t]$,
  - *either* $C^1_{u,k}[|t_s - t_r| + t]$ *is of the form* (C_{add}, C_{join}) *or* C^1_{add} *and* $C^1_{u,\ell}[t] = C_{add}$, $k, \ell \in \{i, j\}, k \neq \ell$,
  - *or* $t = 0$ *and* $C^1_{u,i}[|t_s - t_r|]$ *is of the form* (C_{add}, C_{join}) *or* C_{add} *and* $C^1_{u,j}[0] = C_{join}$ *with* (C_{add}, C_{join}) *having been produced by the same* Add&Join *call.*

We define chains of partnered sessions, as for the RDM, to connect sessions that are active on different devices that were present at different steps. Those structures are necessary to link any session to the initiation step when the authentication is performed.

Definition 4.6 (Chained sessions.)**.** *A session* (u, j, r, t_r) *is chained with* (u, i, s, t_s) *if* t_r *is maximal and there exists* n *sessions* (u, i_α, s_α)*, and* n *couples* (t'_α, t_α)*,* $t'_\alpha \leq t_\alpha$*,* $\alpha \in [0, n-1]$ *such that:*

- (u, j, r, t_r) *and* (u, i_0, s_0, t'_0) *are partnered,*
- $\forall \alpha \in [0, n-2]$*,* $(u, i_\alpha, s_\alpha, t_\alpha)$ *and* $(u, i_{\alpha+1}, s_{\alpha+1}, t'_{\alpha+1})$ *are partnered,*
- $(u, i_{n-1}, s_{n-1}, t_{n-1})$ *and* (i, s, t_s) *are partnered.*

$\{(u, i_\alpha, s_\alpha, t_\alpha)\}_{\alpha \in [0,n-1]}$ *is called a chain of sessions between* (u, j, r, t_r) *and* (u, i, s, t_s)*.*

We consider matching sessions relatively to the peer's conversation. We define $\pi^s_{u,i}.\mathtt{sid_2}$ as the concatenation of ciphertexts c_{out} received by $\mathsf{ReceiveOut}(\pi^{\mathsf{s}}_{\mathsf{u,i}}, \cdot)$ or produced either by $\pi^s_{u,i}$ or by any partnered session. The matching is defined between two sessions (u, i, s) and (v, ℓ, o). One session (u, i, s) can match several other sessions (v, ℓ_z, o_z). Our definition is recursive: a matching is well-defined if one can trace the conversations from the very beginning on u and v's side. This is done by calling chains of sessions, and each chain element should match an element in the other chain.

Definition 4.7 (Matching sessions at some step.)**.** *Two sessions* (u, i, s, t_s) *and* (v, ℓ, p, t_p)*,* $t_s \geq t_p$ *are matching if:*

- $\pi^s_{u,i}.role \neq \pi^p_{v,\ell}.role$,
- $\pi^s_{u,i}.peer = \pi^p_{v,\ell}.user$ *and* $\pi^s_{u,i}.user = \pi^p_{v,\ell}.peer$,
- (u, i, s, t_s) *and* (v, ℓ, p, t_p) *are chained with respective initial sessions* (u, i_0, s_0, t_{s_0})*,* (v, l_0, p_0, t_{p_0})*, through respective chains* $\{(u, i_\alpha, s_\alpha, t_\alpha)\}_{\alpha \in [1,n]}$*,* $\{(v, \ell_\beta, p_\beta, t_\beta)\}_{\beta \in [1,m]}$*,*

- $\exists$ $\mathtt{sid}$ *subset of* $\pi^s_{u,i}[t_s].\mathtt{sid}_2$ *such that* $\pi^s_{u,i}[t_s].\mathtt{sid}_2 = \mathtt{sid} \| \pi^p_{v,\ell}[t_p].\mathtt{sid}_2$,
- *if* (u, i, s, t_s) *is an initial session, then* $\forall\ \beta \in [0, m]$, $\exists\ \tilde{t}_\beta$ *and* $\mathtt{sid}_\beta$ *such that* $\pi^s_{u,i}[\tilde{t}_\beta].\mathtt{sid}_2 = \mathtt{sid}_\beta \| \pi^{p_\beta}_{v,\ell_\beta}[t_\beta].\mathtt{sid}_2$,
- *else* (u, i, s, t_s) *and* (v, ℓ_m, p_m, t_m) *are matching.*

We can estimate $\tilde{t}_\beta = \tilde{t}_{\beta-1} + t_\beta - t'_\beta$, t'_β *defined as in 4.6.*

A multi device messaging solution shall ensure that any device of Alice can receive a message sent by any device of Bob. As we consider the key exchange point of view of Signal, this means that any two matching sessions (one on Alice's side, the other on Bob's side) shall share a same session key if they are matching. Hence we define correctness of a MDIM as follow:

Definition 4.8 (Correctness of a MDIM)**.** *A MDIM protocol is said to be correct if, for all matching sessions* (u, i, s, t_s) *and* (v, j, r, t_r), $\pi^s_{u,i}[t_s].sessionKey = \pi^r_{v,j}[t_r].sessionKey$.

The adversary's power. As we expect our protocol to provide at least as much security as the one-to-one Signal protocol, the adversary shall have - at least - as much power. Hence, in addition to the expected oracles that enables him to run the protocol, we provide $\mathcal{A}$ with oracles RevealUserLT and RevealEphemeralKey that reveal the user long term key (respectively the ephemeral keying material) of the prekey bundle, RevealState, RevealRandom, and RevealSessionKey. To ensure that the presence of multiple devices do not impoverish the security, he is also given oracles to access devices specific data: RevealDevLT, RevealDevState.

MDIM **indistinguishability.** The MDIM-IND complete experiment is described in Figure 4.9. We complete a session state $\pi^s_{u,i}$ with flags linked to different oracles. All flags are initialized with false. Flag `active`, is set to $true$ as soon as $\mathsf{OSend}(\pi^{\mathsf{s}}_{\mathsf{u,i}})$ is called. In a general way, `revX` is set to $true$ whenever revealX is invoked on this session. A flag value is transmitted from one step to another.

We detail below the different registers and counters we need to define our freshness.

- The register $V^s_{u,i}$, as in RDM model in subsection 4.3.2, is there to prevent $\mathcal{A}$ from interfering if a device randomness is corrupted.
- The counter $E^s_{u,i}$ records the step of the last Send.
- The counter $F^s_{u,i}$ records the step when the last change of randomness (what we call ratchet) occurred.
- The counter $G^s_{u,i}$ records the step when the last change of state occurred. $E^s_{u,i}$, $F^s_{u,i}$ and $G^s_{u,i}$ are useful to turn all necessary flags to $true$ when Reveal queries occur. If the Reveal query happens, then we turn all flags from $E^s_{u,i}$ (resp. $F^s_{u,i}$, $G^s_{u,i}$) to $true$. Some actions as Send or Add&Join or Revoke may turn them back to $false$. This corresponds to the healing property. Remark that in OSend and OReceiveIn, we identify ratchets by comparing random values. In OReceiveOut, we identify on state changes by comparing state values.
- The list $A^s_{u,i}$ records all steps when an addition of a new device occurs. We need to keep track of these because information is sent to the newcomer. This list is emptied with every revocation.

For readability reasons we define:
$\mathrm{ResetRand-State-Session(u, i, s, t)}$=
$\pi^s_{u,i}[T]$.revRand $\leftarrow false$ and $\pi^s_{u,i}[T]$.revState $\leftarrow false$ and $\pi^s_{u,i}[T]$.revSessionKey $\leftarrow false$.

$\mathrm{ResetState-Session(u, i, s, t)}$=
$\pi^s_{u,i}[T]$.revState $\leftarrow false$ and $\pi^s_{u,i}[T]$.revSessionKey $\leftarrow false$.

$\mathrm{NoDevReveal(u, i, s, t)}$=
$\neg[\pi^s_{u,i}[t]$.revDevState $\vee(\pi_{u,i}$.revDevLT $\wedge \neg\pi^s_{u,i}[t]$.active$)]$ and
$\forall(u, j, r, t_r)$ chained with (u, i, s, t) $\neg[\pi^r_{u,j}[t_r]$.revDevState$\vee(\pi_{u,j}$.revDevLT$\wedge\neg\pi^r_{u,j}[t_r]$.active$)]$.

Freshness conditions. Freshness conditions are obtained by considering the RDM freshness and upgrading the original Signal freshness in the following way: each time an element of a session was concerned in the original freshness, the same element has now to be considered for this session and all the partnered and chained sessions. More precisely, we try to stipulate clearly all the ways that an element can leak to the adversary: directly from the targeted device or a partnering or matching one, or through the communication between devices. For the latter, data of a session (u, i, s) at step t can leak if a device (u, i)'s randomness (or device keys if it has not been active in the ratcheting process yet) is compromised, or if the device randomness of a partnered session is compromised or if there exists a session chained with (u, i, s) whose device randomness is compromised (if it is chained, it means it will not perform any action until it matches (u, i, s, t) - if not revoked - it just has not received all of its messages). We define:
DeviceLeak(u, i, s, t_s) =

- $\pi^s_{u,i}[t_s]$.revDevState
- or $(\pi_{u,i}$.revDevLT $\wedge \neg\pi^s_{u,i}[t_s]$.active$)$
- or $\exists\ (u, j, r, t_r)$ partnering (u, i, s, t_s) such that:
  - $\pi^r_{u,j}[t_r]$.revDevState
  - or $(\pi_{u,j}$.revDevLT $\wedge \neg\pi^r_{u,j}[t_r]$.active$)$
- or $\exists\ (u, j, r, t_r)$ chained with (u, i, s, t_s) such that:
  - $\pi_{u,j}.pk \in \pi^s_{u,i}[t_s].devices$ and
    - $\pi^r_{u,j}[t_r]$.revDevState
    - or $(\pi_{u,j}.revDevLT \wedge \neg\pi^r_{u,j}[t_r]$.active$)$.

Initial freshness. As in the original model for Signal, freshness of the initial non interactive key exchange (NIKE) is treated separately, because it is proper to X3DH. (We consider here freshness conditions designed for X3DH *i.e.* without the optional one time prekeys. Taking into account the X4DH option would add an oracle on the optional Signal keys and corresponding restrictions.) One can imagine adopting another NIKE and designing a new ad hoc initiation freshness. A major impact of the multi device on this initiation freshness is due to the sharing of the user keys between the devices. The security of the session initiation is now related to the security of the communication between the devices. And the revocations will define intervals of steps within which the security of an initiation shall be considered. Let I be an interval of protocol steps. For readability reasons we define:

DevLeakPrekeys(u, i, s, I) = there exists a step $t \in A^s_{u,i} \cap I$ such that DeviceLeak(u, i, s, t).

$\mathrm{CorruptUser}(u, I)$ = u.`revUserLT` within the period I or there exists a session (u, i, s) such that $\mathrm{DevLeakPrekeys}(u, i, s, I)$.

$\mathrm{RevealEph}(u, I)$ = u.`revEph` within the period I or there exists a session (u, i, s) such that $\mathrm{DevLeakPrekeys}(u, i, s, I)$.

For a session (u, i, s), let I_u identify the interval that starts with the last revocation on u's side before (u, i, s) is initialized and ends with the first revocation on u's side after (u, i, s) is initialized. If no revocation occurs after initialization, I_u ends with the experiment. If no revocation had occurred before the initialization, I_u starts with the experiment. The same way, define $I_{\bar{u}}$ as the interval that starts with the last revocation on $\pi^s_{u,i}.peer$'s side before (u, i, s) is initialized and ends with the first revocation on $\pi^s_{u,i}.peer$'s side after (u, i, s) is initialized.

Definition 4.9 (Initiation freshness.). *Consider a session (u, i, s). A session $\pi^s_{u,i}$ such that $\pi^s_{u,i}.role = initiator$ has a fresh initiation if:*

- $\neg\mathrm{CorruptUser}(u, I_u) \vee \neg\mathrm{RevealEph}(\pi^s_{u,i}.peer, I_{\bar{u}})$ *or*
- $\neg\pi^s_{u,i}[0].\texttt{revRand} \vee \neg\mathrm{RevealEph}(\pi^s_{u,i}.peer, I_{\bar{u}})$ *or*
- $\neg\pi^s_{u,i}[0].\texttt{revRand} \vee \neg\mathrm{CorruptUser}(\pi^s_{u,i}.peer, I_{\bar{u}})$.

A session $\pi^s_{u,i}$ such that $\pi^s_{u,i}.role = responder$ has a fresh initiation if:

- $\neg\mathrm{CorruptUser}(\pi^s_{u,i}.peer, I_{\bar{u}}) \vee \neg\mathrm{RevealEph}(u, I_u)$ *or*
- $\neg\pi^o_{v,k}[0].\texttt{revRand}$ *for all* $(v, k, o, 0)$ *matching* $(u, i, s, 0)$ *if it exists* $\vee \neg\mathrm{RevealEph(u, I_u)}$ *or*
- $\neg\pi^o_{v,k}[0].\texttt{revRand}$ *for all* $(v, k, o, 0)$ *matching* $(u, i, s, 0)$ *if it exists* $\vee \neg\mathrm{CorruptUser(u, I_u)}$.

Freshness of the following steps. Here we consider the same restrictions as in [CCD+17] but we extend them relatively to partnered sessions, multiple matching sessions and device leakage for data that are transmitted between the devices (randomness and state data when a device is added).

Definition 4.10 (Freshness at some step.). *A session (u, i, s, t), $t > 0$, is fresh if:*

- $\neg(\pi_{u,i}.\texttt{revDevLT} \wedge \neg\pi^s_{u,i}[t].\texttt{active})$ *and*
- $\neg\mathrm{RevealSessionKey}$*(u,i,s,t) and*
- $\neg\mathrm{RevealState}$*(u,i,s,t − 1) or* $\neg\mathrm{RevealRandom}(u, i, s, t)$.

where $\mathrm{RevealSessionKey}(u, i, s, t)$ *stands for:*

- $\pi^s_{u,i}[t].\texttt{revSessionKey}$
- *or* $\exists$ *(u,j,r,t_r) partnered with (u,i,s,t) such that* $\pi^r_{u,j}[t_r].\texttt{revSessionKey}$
- *or* $\exists$ *(v,ℓ,o,t_o) matching (u,i,s,t) such that* $\pi^o_{v,\ell}[t_o].\texttt{revSessionKey}$.

$\mathrm{RevealState}(u, i, s, t)$ *stands for:*

- $\pi^s_{u,i}[t].\texttt{revState}$
- *or* $\exists$ *(u,j,r,t_r) partnered with (u,i,s,t) such that* $\pi^r_{u,j}[t_r].\texttt{revState}$
- *or* $\exists$ *(v,ℓ,o,t_o) matching (u,i,s,t) such that* $\pi^o_{v,\ell}[t_o].\texttt{revState}$

$Exp^{\text{MDIM-IND}}_{\mathcal{A},\text{MDIM},n_p,n_d}(\lambda)$

$b \leftarrow\$ \{0,1\}$
$chal \leftarrow \bot, initialdata \leftarrow \bot$
for $u = 1, \ldots, n_p$ **do**
  $P_u \leftarrow \bot$
  for $i = 1, \ldots, n_d$ **do**
    $s_{u,i} \leftarrow 0$
    $dpk_{u,i}, \pi_{u,i} \leftarrow \mathsf{DeviceSetup}(1^\lambda, (u,i))$
    $\pi_{u,i}, pubprekeys_u \leftarrow \mathsf{Register}(id_u, id_i)$
    $initialdata += u, pubprekeys_u, dpk_{u,i}$
$b' \leftarrow \mathcal{A}^{Oracles, Challenge}(initialdata)$
return $chal \neq \bot \wedge Fresh(chal) \wedge b = b'$

OInitSession($role$, u, i, v)

$s_{u,i} \leftarrow s_{u,i} + 1, s \leftarrow s_{u,i}$
$C_{init}, c_{out}, \pi^s_{u,i}[0]$
  $\leftarrow \mathsf{InitSession}(role, \pi_v.pubprekeys, \pi_{u,i}, s)$
$P_u \leftarrow P_u \cup C_{init}$
$\pi_{u,i}.Sessions \leftarrow s, \pi^s_{u,i}.\texttt{step} \leftarrow 0$
$V^s_{u,i} \leftarrow \emptyset$
$E^s_{u,i} \leftarrow 0,\ F^s_{u,i} \leftarrow 0,$
$G^s_{u,i} \leftarrow 0,\ A^s_{u,i} \leftarrow \{0\}$
return c_{out}, C_{init}

OReceiveInitSession(C_{init}, u, j)

$s_{u,j} \leftarrow s_{u,j} + 1, r \leftarrow s_{u,j}$
$\pi^r_{u,j} \leftarrow \mathsf{ReceiveInitSession}(C_{init}, \pi_{u,j}, r)$
$\pi_{u,j}.sessions \leftarrow r,\ \pi^r_{u,j}.\texttt{step} \leftarrow 0$
$V^r_{u,j} \leftarrow \emptyset,\ A^r_{u,j} \leftarrow \{0\}$
$E^r_{u,j} \leftarrow 0,\ F^r_{u,j} \leftarrow 0,\ G^r_{u,j} \leftarrow 0$
return

OSend(m, u, i, s)

$t \leftarrow \pi^s_{u,i}.\texttt{step}$
$C_{in}, c_{out}, \pi^s_{u,i}[t+1] \leftarrow \mathsf{Send}(m, \pi^s_{u,i}[t])$
$V^s_{u,i} \leftarrow V^s_{u,i} \cup C_{in},\ E^s_{u,i} \leftarrow t+1$
for $T \geq t+1$ **do**
  $\pi^s_{u,i}[T].\texttt{revDevRand} \leftarrow false$
if $\pi^s_{u,i}[t].\texttt{active} = false$ **then**
  for $T \geq t+1$ **do**
    $\pi^s_{u,i}[T].\texttt{active} \leftarrow true$
if $\pi^s_{u,i}[t+1].rand \neq \pi^s_{u,i}[t].rand$ **then**
  $F^s_{u,i} \leftarrow t+1, G^s_{u,i} \leftarrow t+1$
  for $T \geq t+1$ **do**
    $ResetRand\text{-}State\text{-}Session(u, i, s, T)$
$\pi^s_{u,i}.\texttt{step} \leftarrow t+1$
return C_{in}, c_{out}

OReceiveIn(C, u, j, r)

$t \leftarrow \pi^r_{u,j}.\texttt{step}$
Req. $NoDevReveal(u, j, r, t)$
  $\vee\ C \in V^r_{u,j}$
  $\vee\ \exists(u, k, o, t_o)$ partnered with (u, j, r, t)
  such that $C \in V^o_{u,k}$
$m, \pi^r_{u,j}[t+1] \leftarrow \mathsf{ReceiveIn}(c, \pi^r_{u,j}[t])$
if $\pi^r_{u,j}[t+1].rand \neq \pi^r_{u,j}[t].rand$ **then**
  $F^r_{u,j} \leftarrow t+1,\ G^r_{u,j} \leftarrow t+1$
  for $T \geq t+1$ **do**
    $ResetRand\text{-}State\text{-}Session(u, j, r, T)$
$\pi^r_{u,j}.\texttt{step} \leftarrow t+1$
return m

OReceiveOut(c, v, ℓ, p)

$t \leftarrow \pi^p_{v,\ell}.\texttt{step}$
$m, \pi^p_{v,\ell}[t+1] \leftarrow \mathsf{ReceiveOut}(c, \pi^p_{v,\ell}[t])$
if $\pi^p_{v,\ell}[t+1].state \neq \pi^p_{v,\ell}.state$ **then**
  $G^p_{v,\ell} \leftarrow t+1$
  for $T \geq t+1$ **do**
    $ResetState\text{-}Session(v, j, r, T)$
$\pi^p_{v,\ell}.\texttt{step} \leftarrow t+1$
return m

OAdd&Join(j, u, i)

$\{C_{join,s}, C_{add,s}, c_{out,s}\}_{s \in \pi_{u,i}.sessions}, \pi_{u,i}$
  $\leftarrow \mathsf{Add\&Join}(pk_j, \pi_{u,i})$
for $s \in \pi_{u,i}.Sessions$ **do**
  $t_s \leftarrow \pi^s_{u,i}.\texttt{step}$
  $P_u \leftarrow P_u \cup \{C_{join,s}\}$
  $V^s_{u,i} \leftarrow V^s_{u,i} \cup \{C_{add,s}\}$
  $F^s_{u,i} \leftarrow t_s + 1,\ G^s_{u,i} \leftarrow t_s + 1$
$A^s_{u,i} \leftarrow A^s_{u,i} \cup \{t_s + 1\}$
  $\pi^s_{u,i}.\texttt{step} \leftarrow t_s + 1$
  for $T \geq t_s + 1$ **do**
    $ResetRand\text{-}State\text{-}Session(u, i, s, T)$
return $\{C_{join,s}, C_{add,s}, c_{out,s}\}_{s \in \pi_{u,i}.Sessions}$

ODecJoin($\{C_r\}_{r \in [1,R]}$, u, j)

for $1 \leq r \leq R$ **do** Req. $C_r \in P_u$
$\pi_{u,j} \leftarrow \mathsf{DecJoin}(\{C_r\}_{r \in [1,R]}, \pi_{u,j}, r)$
$s_{u,j} \leftarrow R$
for $1 \leq r \leq R$ **do**
  $E^r_{u,j} \leftarrow 0,\ F^r_{u,j} \leftarrow 0,\ G^r_{u,j} \leftarrow 0$
  $V^r_{u,j} \leftarrow \bot, A^r_{u,j} \leftarrow \{0\},\ \pi^r_{u,j}.step \leftarrow 0$
return

Figure 4.9 – The Multi-Device Ratcheted Key Exchange Model.

ODecAdd($\{C_o\}_{o\in[1,O]}$, u, k)

1 : Req. $O = \#\pi_{u,k}.Sessions$
2 : **for** $1 \le o \le O$ **do**
3 : $t_o \leftarrow \pi^o_{u,k}.step$
4 : Req. $NoDevReveal(u,k,o,t_o)$
$\vee\ C_o \in V^o_{u,k}$
$\vee\ \exists(u,i,s,t_s)$ partnered with (u,k,o,t_o) such that $C_o \in V^s_{u,i}$
5 : $\pi_{u,k} \leftarrow \mathsf{DecAdd}(\{C_o\}_{o\in[1,O]}, \pi_{u,k})$
6 : **for** $1 \le o \le O$ **do**
7 : **for** $T \ge t_o + 1$ **do**
8 : $ResetRand\text{-}State\text{-}Session(u,k,o,T)$
9 : $F^o_{u,k} \leftarrow t_o + 1,\ G^o_{u,k} \leftarrow t_o + 1$
10 : $A^o_{u,k} \leftarrow A^o_{u,k} \cup \{t_o\}$
11 : $\pi^o_{u,k}.step \leftarrow t_o + 1$
12 : **return**

ORevoke(dpk, u, i)

1 : $\{C_{rev,s}, c_{out,s}\}_{s\in\pi_{u,i}.Sessions}, \pi_{u,i},$
$pubprekeys \leftarrow \mathsf{Revoke}(dpk, \pi_{u,i})$
2 : **for** $s \in \pi_{u,i}.Sessions$ **do**
3 : $t_s \leftarrow \pi^s_{u,i}.\mathtt{step}$
4 : $V^s_{u,i} \leftarrow V^s_{u,i} \cup \{C_{rev,s}\}$
5 : $F^s_{u,i} \leftarrow t_s + 1,\ G^s_{u,i} \leftarrow t_s + 1,\ A^s_{u,i} \leftarrow \emptyset$
6 : $\pi^s_{u,i}.\mathtt{step} \leftarrow t_s + 1$
7 : **for** $T \ge t_s + 1$ **do**
$ResetRand\text{-}State\text{-}Session(u,i,s,T)$
8 : $u.\mathtt{corruptUser} \leftarrow false$
9 : $u.\mathtt{corruptOpt} \leftarrow false$
10 : **return** $\{C_{rev,s}, c_{out,s}\}_{s\in\pi_{u,i}.Sessions}, pubprekeys$

ODecRevoke($\{C_o\}_{o\in[1,O]}$, u, k)

1 : Req. $O = \#\pi_{u,k}.Sessions$
2 : **for** $1 \le o \le O$ **do**
3 : $t_o \leftarrow \pi^o_{u,k}.\mathtt{step}$
4 : Req. $NoDevReveal(u,k,o,t_o)$
$\vee\ C_o \in V^o_{u,k}$
$\vee\ \exists(u,i,s,t_s)$ partnered with (u,k,o,t_o) such that $C_o \in V^s_{u,i}$
5 : $\pi_{u,k} \leftarrow \mathsf{DecRevoke}(\{C_o\}_{o\in[1,O]}, \pi_{u,k})$
6 : **for** $1 \le o \le O$ **do**
7 : **for** $T \ge t_o + 1$ **do**
8 : $ResetRand\text{-}State\text{-}Session(u,k,o,T)$
9 : $F^o_{u,k} \leftarrow t_o + 1,\ A^o_{u,k} \leftarrow \emptyset$
10 : $\pi^o_{u,k}.\mathtt{step} \leftarrow t_o + 1$
11 : **return**

RevealDevState(u, i, s)

1 : $t \leftarrow \pi^s_{u,i}.step$
2 : **for** $T \ge E^s_{u,i}$ **do**
3 : $\pi^s_{u,i}[T].\mathtt{revDevState} \leftarrow true$
4 : **if** $\neg\pi^s_{u,i}[t].\mathtt{active}$ **then**
5 : $\pi_{u,i}.\mathtt{revDevLT} \leftarrow true$
6 : **return** $\pi^s_{u,i}[t].devrand$

RevealSessionKey(u, i, s)

1 : $t \leftarrow \pi^s_{u,i}.step$
2 : **for** $T \ge t$ **do**
3 : $\pi^s_{u,i}[T].\mathtt{revSessionKey} \leftarrow true$
4 : **return** $\pi^s_{u,i}[t].sessionkey$

RevealState(u, i, s)

1 : $t \leftarrow \pi^s_{u,i}.step$
2 : **for** $T \ge G^s_{u,i}$ **do**
3 : $\pi^s_{u,i}[T].\mathtt{revState} \leftarrow true$
4 : **return** $\pi^s_{u,i}[t].state$

RevealRandom(u, i, s)

1 : **for** $T \ge F^s_{u,i}$ **do**
2 : $\pi^s_{u,i}[T].\mathtt{revRand} \leftarrow true$
3 : **return** $\pi^s_{u,i}[t].Random$

RevealDevLT(u, i)

1 : $\pi_{u,i}.\mathtt{revDevLT} \leftarrow \mathsf{true}$
2 : **return** $\pi_{u,i}.dsk_{u,i}$

RevealUserLT(u)

1 : $u.\mathtt{revUserLT} \leftarrow \mathsf{true}$
2 : **return** $\pi_u.usk_u$

RevealEphemeral(u)

1 : $u.\mathtt{revEph} \leftarrow true$
2 : **return** $\pi_u.mtsk_u$

Challenge(u, i, s, t)

1 : $chal \leftarrow (u,i,s,t)$
2 : **if** $b = 0$ $k \leftarrow \pi^s_{u,i}[t].sessionkey$
3 : **else** $k \leftarrow\!\!\$\ \mathcal{K}$
4 : **return** k

- *or* $t \in A^s_{u,i}$ *and* $\mathrm{DeviceLeak}(u, i, s, t)$
- *or* $\exists$ *(v,k,o,*t_o*) matching (u,i,s,t) such that* $t_o \in A^o_{v,k}$ *and* $\mathrm{DeviceLeak}(v, k, o, t_o)$

$\mathrm{RevealRandom}(u, i, s, t)$ *stands for:*

- $\pi^s_{u,i}[t]$.`revRand`
- *or* $\exists$ *(u,j,r,*t_r*) partnering (u,i,s,t) such that* $\pi^r_{u,j}[t_r]$.`revRand`
- *or* $\mathrm{DeviceLeak}(u, i, s, t)$*.*

PFS, revocation and out-of-order messages. As for RDM, we do not consider out-of-order messages. We have already mentionned that this option, only considered in the work of Alwen *et al.* , seriously damages the PFS. The other fundamental reason why we made this choice is the revocation feature. If Bob still accepts unused old keys to face up with the arrival of delayed messages, a revoked device of Alice can also use these keys to infiltrate maliciously the session. Revocation would not be efficient. The second obstacle to forward secrecy in a multi-device context is that we consider each device shall receive all the messages in the conversation. If a device stays offline for a long time, it will process all the updates from the moment he went offline until the moment he is back online. All the corresponding keys are still sensible data. This is why we need to consider chains of session in the freshness conditions. Forward secrecy is to be considered only for the messages sent before the "oldest offline device" went offline. This highlights that a multi-device application should consider a process to prevent the devices from being offline for a time too long (automatic revocation for instance). This is to be considered at an implementation level.

Definition 4.11 (Secure Multi-Device Instant Messaging.)**.** *A* MDIM *executed with* n_p *users, each having* n_d *devices is said to be secure in the above model if it is correct and for all adversary* $\mathcal{A}$ *running in polynomial time, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\mathsf{Adv}^{\text{MDIM-IND}}_{\mathcal{A},\text{MDIM},n_p,n_d}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\text{MDIM-IND}}_{\mathcal{A},\text{MDIM},n_p,n_d}(\lambda)\right] - \frac{1}{2}\right|.$$

4.4.3 Building over Signal

We detail our Multi-Device Signal solution, depicted in Figure 4.4. It is built from the Signal protocol and our RDM protocol described in subsection 4.3.3. The complete pseudocode description is given in Figure 4.11. The RDM enables us to share the DH secrets, for all devices to perform the operation. It is also used to share the message's body. Every Signal sending is doubled with a RDM sending. This way, any device can follow the conversation, can speak for itself, and can directly receive messages sent by Bob. For each Signal session, a specific RDM channel is opened between Alice's devices. Addition and revocation induce extra ratchets, for the joining/revoked device not to access previous/future conversations. We introduce a new procedure, `ExtraRatchet`. When a device receives a ratchet secret through the RDM channel, he has to update its Signal state accordingly. This is done in an Update procedure. Those new procedures are detailed in section 4.4.3.

Signal. We describe Signal, that we depicted in Figure 3.5 with the five following algorithms:

- $\mathsf{Sig.KeyGen}(1^\lambda) \to pk, sk$.
- $\mathsf{Sig.MedTermKeyGen}(1^\lambda) \to ephpk, ephsk$.

- Sig.Activate$(role, ephpk, \pi^s_{u,i}) \rightarrow c_{out}, \pi^s_{u,i}$. Computes the initial shared secret, the first rootkey rk and the first chainkey ck. Returns a message c_{out} and an updated state $\pi^s_{u,i}$.

- Sig.Send$(m, \pi^s_{u,i}) \rightarrow c_{out}, \pi^s_{u,i}$. A probabilistic algorithm that takes as input a session state $\pi^s_{u,i}$ and a message m, and returns an updated state $\pi^s_{u,i}$ and a ciphertext c.

- Sig.Receive$(c, \pi^s_{u,i}) \rightarrow m, \pi^s_{u,i}$. A deterministic algorithm that takes as input a session state $\pi^s_{u,i}$ and a ciphertext c, and returns an updated state $\pi^s_{u,i}$ and a message m.

This corresponds to the abstraction adopted in [CCD+17], except that we split the algorithm Run into Sig.Send and Sig.Receive, which corresponds to the RKE abstraction from section 3.3. We need to keep the key generation and activation algorithms detail as, contrary to the RKE formalisation, we do take into account the initial non interactive key exchange. We detail Sig.Register to take into account the device key in addition to the user key and the ephemeral keys usually used by Signal.

The extra procedures. A device shall perform a RDM.SetUp to obtain its devices keys before he registers. We gather Sig.KeyGen and Sig.MedTermKeyGen in a UserKeyGen procedure that returns a set of prekeys $prekeys$. Those keys are registered to the server. We obtain a Multi-Device Instant Messaging protocol, as formally defined in subsection 4.4.1. We formalize the registering procedure as follow:

- Sig.Register$(uID, \pi_{u,i}, pprekeys) \rightarrow \pi_{u,i}$. On input a user ID, a device state and a set of prekeys $prekeys$, registers on the server and updates the device state $\pi_{u,i}$ with Signal data.

We detail in Figure 4.10 below the ExtraRatchet and Update procedures. The first is needed to ensure confidentiality of conversation before adding a device or after revoking one. The second enables devices that receive a ratchet secret through the RDM channel (in a ReceiveIn), to maintain their Signal state up-to-date.

Update$(\pi^s_{u,i}, rchsk)$	ExtraRatchet$(\pi^s_{u,i})$
1 : **if** $rchsk \neq \pi^s_{u,i}.rand$	1 : $rchsk, rchpk \leftarrow DHKeyGen(1^\lambda)$
2 : $\quad E \leftarrow DH(rchsk, \pi^s_{u,i}.rand_{peer})$	2 : $E \leftarrow DH(rchsk, \pi^s_{u,i}.rand_{peer})$
3 : $\quad RK, CK \leftarrow KDF\_RK(\pi^s_{u,i}.RK, E)$	3 : $RK, CK \leftarrow KDF\_RK(\pi^s_{u,i}.RK, E)$
4 : $\quad \pi_{u,i}s.RK \leftarrow RK, \pi^s_{u,i}.CK \leftarrow CK$	4 : $\pi^s_{u,i}.RK \leftarrow RK, \pi^s_{u,i}.CK \leftarrow CK$
5 : $CK, MK \leftarrow\leftarrow KDF\_CK(CK)$	5 : $\pi^s_{u,i}.rand \leftarrow rchsk$
6 : $\pi^s_{u,i}.CK \leftarrow CK, \pi^s_{u,i}.MK \leftarrow MK$	6 : **return** $rchpk, \pi^s_{u,i}$
7 : **return** $\pi^s_{u,i}$	

Figure 4.10 – The ExtraRatchet and Update procedures.

About addition and revocation. Our Add&Join sends $sprekeys$ and $Devices$ to everybody (the newcomer as the already enrolled devices). This last point is done on purpose to be sure a newcomer cannot receive session specific information without receiving global ones. Sending these data only with the RDM.Join would lead to a more complicated model for the RDM and we choose to keep the joining action unrelated to the shipping of encrypted messages.

The complete construction. We give in Figure 4.11 a complete pseudo code description of our Multi-Device Signal protocol, MDSig. As all Signal and RDM procedures take as an entry the device state and update it, we simplify (except for initiation and key generation procedures) $\pi^s_{u,i}, y \leftarrow \mathsf{Proc}(\pi^s_{u,i}, x)$ as $y \leftarrow \mathsf{Proc}(x)$.

The following theorem states the security of the above construction:

Theorem 4.2. *Let Signal be a secure multi-stage key-exchange protocol with advantage ϵ_{sig} and* RDM *a* RDM-IND *secure ratcheted dynamic multicast with advantage $\epsilon_{\text{RDM-IND}}$, the above construction is a secure* MDIM *such that, for any PPT adversary running n_s sessions from n_d devices of n_p users, making at most q queries to the oracles:*

$$\mathsf{Adv}^{\text{MDIM-IND}}_{\mathcal{A},\mathsf{MDSig},n_p,n_d}(\lambda) \leq n_p^2 \cdot (2 \cdot \epsilon_{\text{RDM-IND}} + \epsilon_{sig}).$$

A proof of the security of our MDIM **construction.**

Proof. Correctness of the partnering. Suppose (u,i,s,t_s) and (u,j,r,t_r) are partnered sessions, with (u,j,r) being alive for longer than (u,i,s) ($t_r > t_s$). That means $\pi^r_{u,j}[t_r].\mathtt{sid_1} \tilde{=} sid' \| \pi^s_{u,i}[t_s].\mathtt{sid1}$. By definition of $\mathtt{sid_1}$, the two sessions are matching in the sense of ratcheted multicast. By the construction we have that $\pi^s_{u,i}[t_s].\mathtt{sid_1}[0]{=}C_{join}$ where C_{join} is obtained from $\mathsf{Add\&Join}(pk_i, \pi^o_{u,k})$ for some session (u,k,o). By the partnering definition, two cases are possible as C_{join} is intended to (u,i,s): either $\pi^r_{u,j}[t_r].\mathtt{sid_1} = sid' \| (C_{join}, C_{add}) \| \cdots$, (meaning that ($u$,$k$,$o$)=($u$,$j$,$r$)) or $\pi^r_{u,j}[t_r].\mathtt{sid_1} = sid' \| C_{add} \| \cdots$ (meaning that (u,k,o) $\neq$ (u,j,r)) and both sessions share the same RK and CK, as they are encrypted in the ciphertext C_{in} included in both C_{join} and C_{add} ciphertexts of the $Add\&Join$ algorithm. From this step, both sessions are included in the group of the multicast and by correctness of the multicast, all messages exchanged via the multicast canal are the same for (u, i, s) and (u, j, r): they have access to the same successive $randsk$. We now consider the chain of session from (u,i,s,t_s) to an initial session (u,i_{init},s_{init},t_{init}). We can see that the same chain links (u,j,r,t_r) to the same initial session. By construction, (u,j,r,t_r) and (u,i,s,t_s) will receive the same Signal message in ReceiveOut hence the same public randomness from the conversation peer. Since they have common root key and then common secret and public ratchet randomness they compute the same $sessionkey$.

Correctness of the matching. Suppose (u,i,s,t_s) and (v,ℓ,p,t_p) are matching sessions. Consider the chains of sessions from (u,i,s) to an initial session (u,i_0, s_0) (chain U) and from (v,ℓ,p) to the initial session (v,ℓ_0,p_0) (chain V). Those two chains exist according to the definition **??**. Now we consider an abstract super device S_u that is present from the initiation step of session (u, i_0, s_0), and until the step t_s of session (u, i, s). Its state is limited to RK, $randsk$ and $\mathtt{sid_2}$. The state of this super device takes successively the values of the state of the different sessions composing the chain U. This is possible without conflict, because, when two partners are present at the same time, they share the same RK, $randsk$ and $\mathtt{sid_2}$. (For $\mathtt{sid_2}$ this is by definition, for the others, by correctness of partnering). $S_u.\mathtt{sid_2}$ contains all Signal messages received and sent from the initialization to the present step t_r of (u, j, r). We consider a similar super device S_v that aggregates state information along the V chain. The recursive definition of the matching ensures those two users are matching in terms of Signal transcripts at each moment, including the initialization step. The correctness of the Signal protocol provides that S_u and S_v share the same session key at each moment. As $S_u.sessionkey$ (respectively $S_v.sessionkey$) is defined as $\pi^s_{u,i}.sessionkey$ (resp. $\pi^s_{u,i}.sessionkey$)

when session (u,i,s) (resp. (v,j,r)) is alive, we obtain that (u,j,i,t_s) and (v,j,r,t_r) share the same $sessionkey$.

MDIM *indistinguishability.* For all games G_x, we denote S_x the event $\mathcal{A}$ wins in G_x.

Game 0 is the original MDIM-IND Game as described in Figure 4.9.

In **Game 1**, we guess which pair of users (initiator/receiver) (u, v) will be targeted by the adversary in order to apply only on them the RDM-IND security. We have n_p users.

$$\mathsf{Adv}^{\mathrm{G}_0}_{\mathcal{A}}(\lambda) \leq n_p^2 \cdot \mathsf{Adv}^{\mathrm{G}_1}_{\mathcal{A}}(\lambda).$$

In **Game 2**, we will collapse all devices of each of these two users into a super single user. First, as the MDIM-IND security game respects the same restrictions as in RDM ind game for the RDM part, RDM security of the multicast ensures none of the information sent through the RDM canal leaks with security loss bounded by the RDM-IND factor. Second, correctness of the RDM ensures us at each step all "alive" devices share same secrets and transcripts for signal part and that the device that sent the signal message is present in the partnering pool. With this consideration, one can think of the pool of devices of one user as a single signal superdevice.

$$|\Pr[S_2] - \Pr[S_1]| \leq 2 \cdot \epsilon_{\text{RDM-IND}}.$$

What we have to show is that these superdevices execute the Signal protocol as a classical device. What is different from the original Signal design? There are additions and revocations of devices. Once the multicast communication are "erased" (swallowed by superdevice), there remains additional asymmetric signal ratchets (not at initial stage) and prekey bundle updates.

About the asymmetric ratchets. Several ratchets in a row on Alice's side for instance means that Bob's ratchet random will be used several times. In the proofgiven in [CCD+17], this would correspond to a [asym-ir or ri] case. There are two options, either freshness relays on the root key, then randoms are not concerned and we are still in the model. Either freshness relays on randoms. In our model, freshness flag on a random is maintained from step to step: if it is corrupted at a step t, it will remain corrupted for the followings steps. So assumptions on the freshness on the root key or on the random still holds. If the tested session is an initiator type session and that there has been several [asym-ir] type step before challenge, there is no problem. We will receive values (X, Y) from the GDH Challenger and replace last initiator public random by X, and last responder public random by Y as much time as needed (the number of ratchet step there has been on the initiator's side). We knows the successive initiator's ratchet secret and can evaluate the root key as needed. If test occurs on responder's side after several ratchet on responder side it is the same. Responder's public key eventually has been used many steps before but we can simulate perfectly the corresponding steps (not concerned with the DDH challenge) as we know initiator's secret in this case. Additional ratchets do not alter Signal's model with GDH assumption.

About the additional keys. The devices keys are only concerned with the multicast. The main difference is that the entire Signal prekey bundle can be refreshed with the revoke algorithm. Note that, in Figure 4.4, we mention the generation and the transmission of the new prekey bundle to the other device in the Revoke algorithm, but we do not formalise the communication of these keys to the server. We consider that this has to be seen at the implementation level. This refresh should correspond to an unregistering then registering again with the difference here that already ongoing sessions are still alive. Indeed, these ongoing sessions continue to be used as an asymmetric ratchet

has already been performed during the revoke algorithm but the Signal prekey bundle related to these sessions is not used anymore (due to the ratchets). For future new session, not initiated yet, the new Signal prekey bundle is used and it can be viewed as a Signal prekey bundle for a new user. In others terms, it is as if each superdevice were composed of many independant Signal users. In Game 2, we are in the traditional Signal protocol execution.

In **Game 3**, in the Challenge, if b=0, we replace the session key by a random. Signal security ensures we can do this substitution and we have: $|\Pr[S_3] - \Pr[S_2]| = \epsilon_{sig}$.

Finally, in Game 3, $\mathcal{A}$ has no advantage since the returned key is always random and we obtain:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathrm{G}_0}(\lambda) \leq n_p^2 \cdot (2 \cdot \epsilon_{\text{RDM-IND}} + \epsilon_{sig}).$$

□

4.5 A proof of concept implementation.

In this section, we give some measures we made on a first implementation. This work intends to show that the modularity of our construction is not only "on paper". We implement our solution over the Signal library libsignal-protocol-java accessible on `https://github.com/signalapp/libsignal-protocol-java`. This is the one library considered in [CCD+17]. We build our test in the experimental InMemory version of the Library. This version does not use a physical server but simulates the transport layout. We use the JCE and the BouncyCastle libraries for cryptography services. We implemented our RDM with $ECIES$ on curve $secp256r1$ and AES with 128 bits keys for hybrid encryption, and $HMAC\_SHA256$ for the MAC scheme. We play a scenario where Alice uses three devices and Bob one. We run n iterations of the following: Alice sends a message from a random device, Bob and Alice other devices decrypt, Bob answers, all devices of Alice decrypt. Finally, to compare our implementation, we run a similar scenario with Alice devices represented by 3 different users in the Sesame solution. In Table 4.1, we presents the results in term of time and number of connection (each time a message is sent or received) for a run of 1 000 exchanges. We run our test on a 2,9 GHz IntelCore i7 with 16 Go of LPDDR3 RAM memory at 2 133 MHz. Our code is accessible on `https://github.com/multidevicerke`. In subsection 4.5.1, we detail how we locally patched the original library.

	time (ms)	number of connections	data weight (bits)
our solution	20 690	8 000	2 186 000
Sesame	3 007	12 000	990 000

Table 4.1 – Results for a run of $n = 1\ 000$ exchanges.

The Sesame solution is quicker which can be explained by the use of asymmetric encryption in the RDM scheme. However, the difference can be minimized since asymmetric like computation in the Sesame version (Diffie Hellman asymmetric ratchet for all channels) are done using a native elliptic curve C library, whereas we employ an external Java BouncyCastle Library for the encryption computations. Finally, our solution requires one-third less connections. This corresponds to Alice sending only one message for Bob over a Signal channel instead of $n_A + n_B - 1$ for Sesame. Counting all the messages, (those from the RDM and the one that goes through the Signal sevrer), our solution requires n_A messages instead of $n_A + n_B - 1$ for Sesame, which respresents, as n_A and n_B can be expected to be of the same order, a saving of half of the connexions. As a connection is an irreducible

time-consuming operation, this gain is not negligible. Considering the amount of exchanged data, we have a ratio of 273 bits/connection which is largely acceptable.

4.5.1 Patches on the Signal implementation.

We add a half − ratchet method that corresponds to our ExtraRatchet procedure described in subsection 4.4.3. In the original code, as soon as Bob receives a reply, he performs two asymmetric step. He computes the ratchet with Alice new randomness and obtains the root key, chain key pair: (RK_r, CK_r) also computed by Alice. Then Bob immediately prepares his keys for his reply. He generates his new randomness and computes a new pair (RK_s, CK_s). RK_s becomes Bob's current root key. CK_s defines Bob's future sending chain. The positive of this solution is that the sending procedure does not have to consider whether to perform an asymmetric ratchet or not. The negative is that Bob stores his future secret keys before it is necessary, which downgrades the future secrecy property. In our solution, Alice can send another ratchet. She uses her current root key, which is equal to RK_r. When Bob tries to computes the ratchet from his current root key RK_s, he fails. We separate the two ratchet step: the receiver chain is updated when receiving a message, and the sending chain is updated, if necessary, before sending a message. We add a `RatchetCounter` in the Signal state to deal with whether or not perform a ratchet in the Encrypt procedure.

DeviceSetUp($1^\lambda, (u,i)$)

$\pi_{u,i} \leftarrow R.SetUp(1^\lambda, (u,i))$
return $\pi_{u,i}$

UserKeyGen(1^λ)

$usk, upk \leftarrow \mathsf{S.KeyGen}(1^\lambda)$
$ephpk, ephsk \leftarrow \mathsf{S.MTKeyGen}(1^\lambda)$
$ppk \leftarrow upk, mtpk, \{opk_\ell\}_\ell$
$spk \leftarrow usk, mtsk, \{osk_\ell\}_\ell$
return ppk, spk

Register(uID, $\pi_{u,i}$)

if uID already registered **then**
  return / need to do add in that case
else
  $ppk, spk \leftarrow \mathsf{UserKeyGen}(1^\lambda)$
  $\pi_{u,i} \xleftarrow{\cup} \mathsf{S.Register}(\mathsf{uID}, \pi_{u,i}, ppk)$
  $\pi_{u,i}.secprekeys \leftarrow spk$
  $\pi_{u,i}.Devices \leftarrow \emptyset$
  $\pi_{u,i}.Sessions \leftarrow \emptyset$
  return $\pi_{u,i}$

InitSession($role, pprekeys_v, \pi_{u,i}$)

$c_{out} \leftarrow \mathsf{S.Activate}(role, opk_\ell)$
$\pi_{u,i}.Sessions \xleftarrow{\cup} \{s\}$
$\pi^s_{u,i} \leftarrow \mathsf{R.Init}(1^\lambda, \pi_{u,i}, s)$
$spk, rchsk, CK, RK \leftarrow \pi^s_{u,i}$
$D \leftarrow \pi^s_{u,i}.Devices$
$C_{join}, C_{add} \leftarrow \mathsf{R.Add\&Join}(D, \pi^s_{u,i})$
$m_{RDM} \leftarrow RK \| CK \| rchsk \| spk$
$C_{in} \leftarrow \mathsf{R.Enc}(m_{RDM}, \pi^s_{u,i})$
$C_{init} \leftarrow C_{join} \| C_{in}$
return $C_{init}, c_{out}, \pi^s_{u,i}$

ReceiveInitSession($C_{init}, \pi_{u,j}$)

$C_{join} \| C_{in} \leftarrow C_{init}$
$\pi^r_{u,j} \leftarrow \mathsf{R.DecJoin}(C_{join}, \pi_{u,j})$
$RK \| CK \| rchsk \leftarrow \mathsf{R.Dec}(C_{in}, \pi^r_{u,j})$
$\pi^r_{u,j}.state \leftarrow RK, CK$
$\pi^r_{u,j}.rand \leftarrow rchsk$
$\pi_{u,j}.Sessions \leftarrow r$
return $\pi^r_{u,j}$

Add&Join($dpk_j, \pi_{u,i}$)

Require $dpk_j \notin \pi_{u,i}.Devices \wedge i \neq j$
$\pi_{u,i}.Devices \xleftarrow{\cup} dpk_j$
$D \leftarrow \pi_{u,i}.Devices$
$spk \leftarrow \pi_{u,i}.sprekeys$
for *session* $s \in \pi_{u,i}.Sessions$ **do**
  $C_{join,s}, C_{add,s} \leftarrow \mathsf{R.Add\&Join}(dpk_j, \pi^s_{u,i})$
  $\mathsf{ExtraRatchet}(\pi^s_{u,i})$
  $c_{out} \leftarrow \mathsf{S.Send}(\text{"}update\text{"}, \pi^s_{u,i})$
  $RK_s, CK_s, rchsk_s \leftarrow \pi^s_{u,i}$
  $m_s \leftarrow RK_s \| CK_s \| rchsk_s \| spk \| D$
  $C_{in,s} \leftarrow \mathsf{R.Enc}(m_s, \pi^s_{u,i})$
  $C_{join,s} \leftarrow C_{join,s} \| C_{in,s}$
  $C_{add,s} \leftarrow C_{add,s} \| C_{in,s}$
return $\{C_{join,s}, C_{add,s}, c_{out,s}\}_{s \in \pi_{u,i}.Sessions}, \pi_{u,i}$

Figure 4.11 – The Multi-Device Signal construction MDSig. $A \xleftarrow{\cup} \{x\}$ stands for $A \leftarrow A \cup \{x\}$

DecAdd$\left(\{C_{add,o}\}_{o\in[1,O]}, \pi_{u,k}\right)$

for *session* $o \in [1, O]$ **do**
  $C_{add,o} \| C_{in,o} \leftarrow C_{add,o}$
  $\mathsf{R.DecAdd}(C_{add,o}, \pi^o_{u,k})$
  $m_o \leftarrow \mathsf{R.Dec}(C_{in,o}, \pi^o_{u,k})$
  $RK_o \| CK_o \| rchsk_o \| spk \| D \leftarrow m$
  $\pi^o_{u,k}.rand \leftarrow rchsk_o$
  $\pi^o_{u,k}.RK \leftarrow RK_o$
  $\pi^o_{u,k}.CK \leftarrow CK_o$
$\pi_{u,k}.Devices \leftarrow D$
return $\pi_{u,k}$

DecJoin$\left(\{C_{join,r}\}_{r\in[1,R]}, \pi_{u,j}\right)$

for $r \in [1, R]$ **do**
  $C_{join,r} \| C_{in,r} \leftarrow C_{join,r}$
  $\mathsf{R.DecJoin}(C_{join,r}, \pi_{u,j}, r)$
  $m_r \leftarrow \mathsf{R.Dec}(C_{in,r}, \pi^r_{u,j})$
  $RK_r \| CK_r \| rchsk_r \| spk \| D \leftarrow m_r$
  $\pi^r_{u,j}.rand \leftarrow rchsk_r$
  $\pi^r_{u,j}.RK \leftarrow RK_r$
  $\pi^r_{u,j}.CK \leftarrow CK_r$
  $\pi_{u,j}.sprekeys \leftarrow spk$
  $\pi_{u,j}.Devices \leftarrow D$
return $\pi_{u,j}$

Send$(m, \pi^s_{u,i})$

$c_{out} \leftarrow \mathsf{S.Send}(m, \pi^s_{u,i})$
$rchsk \leftarrow \pi^s_{u,i}$
$C_{in} \leftarrow \mathsf{R.Enc}(rchsk \| m, \pi^s_{u,i})$
return $C_{in}, c_{out}, \pi^s_{u,i}$

ReceiveOut$(c_{out}, \pi^p_{v,\ell})$

$m \leftarrow \mathsf{S.Receive}(c_{out}, \pi^p_{v,\ell})$
return $m, \pi^p_{v,\ell}$

ReceiveIn$(C_{in}, \pi^r_{u,j})$

$m, rchsk \leftarrow \mathsf{R.Dec}(C_{in}, \pi^r_{u,i})$
$\mathsf{Update}(\pi^r_{u,j}, rchsk)$
$\pi^r_{u,j}.rand \leftarrow rchsk$
return $m, \pi^r_{u,j}$

Revoke$(devpk, \pi_{u,i})$

Require $devpk \neq \pi_{u,i}.dpk$
Find j s.t. $\pi_{u,i}.Devices[j] = devpk$
if $j = \perp$ **return** $\perp, \pi_{u,i}$
$ppk, spk \leftarrow \mathsf{UserKeyGen}(1^\lambda)$
$S \leftarrow \pi_{u,i}.Sessions$
for s *in* S **do**
  $pk_{j,s} \leftarrow \pi^s_{u,i}.PK[j]$
  $rchsk_s \leftarrow \pi^s_{u,i}.rand$
  $C_{rev,s} \leftarrow \mathsf{R.Revoke}(pk_{j,s}, \pi^s_{u,i})$
  $\mathsf{ExtraRatchet}(\pi^s_{u,i})$
  $c_{out,s} \leftarrow \mathsf{S.Send}(\text{“”}, \pi^s_{u,i})$
  $m_{RDM} \leftarrow rchsk_s \| spk \| devpk$
  $C_{in,s} \leftarrow \mathsf{R.Enc}(m_{RDM}, \pi^s_{u,i})$
  $C_{rev,s} \leftarrow C_{rev,s} \| C_{in,s}$
$\pi_{u,i}.Devices \overset{\setminus}{\leftarrow} \{devpk\}$
return $\{C_{rev,s}, c_{out,s}\}_{s\in S}, ppk, \pi_{u,i}$

DecRevoke$(\{C_{rev,o}\}_{o\in\pi_{u,k}.Sessions}, \pi_{u,k})$

for *session* $o \in \pi_{u,k}.Sessions$ **do**
  $C_{rev,o} \| C_{in,o} \leftarrow C_{rev,o}$
  $\mathsf{R.DecRevoke}(C_{rev,o}, \pi^o_{u,k})$
  $m_o \leftarrow \mathsf{R.Dec}(C_{in,o}, \pi^o_{u,k})$
  $rchsk_o \| spk \| devpk \leftarrow m_o$
  $\pi^o_{u,k}.rand \leftarrow rchsk_o$
  $\pi_{u,k}.sprekeys \leftarrow spk$
  $\pi_{u,k}.Devices \overset{\setminus}{\leftarrow} \{devpk\}$
return $\pi_{u,k}$

and $A \overset{\setminus}{\leftarrow} \{x\}$ stands for $A \leftarrow A \setminus \{x\}$

From One-to-One to Group Instant Secure Messaging

5

Most of the time, Instant Messaging applications are not used for one-to-one communications. One of their most appreciated functionality is that they enable to create groups of users, giving birth to virtual communities. However, a great number of applications base their end-to-end security on the Double Ratchet algorithm which is, as we have studied in chapter 3, dedicated to one-to-one communications. As a consequence, as for the multi-device setting, other solutions, such as the sender's key protocol (*cf.* section 4.1) have been adopted. Those alternatives do not provide the same security level, especially on the PCS aspect. In this chapter, we focus on the Messaging Layer Security (MLS) protocol, which is currently developed by the Internet Engineering Task Force (IETF) and aims at providing a secure group messaging solution. We develop on an identified weakness of this protocol and propose a solution to overcome it. The contributions exposed in this chapter were presented at ESORICS 2021 [DDF21].

Contents

5.1 Messaging Layer Security

MLS targets secure *group* messaging and is developed by the IETF[1]. The goal is to obtain similar security properties as those in one-to-one protocols. The idea is to enable a group of users to share a common secret that can be updated regularly by any member. One of the open issues in the IETF draft is that the validity of an update message can only be checked *after* it has been received. This open issue is clearly identified in the current draft 11 ([BBM+20], section 15.5). Our contribution aims at resolving this issue. However, before diving into the detailed description of the protocol, we give a short overview of the existing work around the global security of the protocol.

5.1.1 A short MLS history

As we have seen in chapter 3, RKE have received a lot of attention in the past decade. Literature for the group version has grown only very recently (mainly from the late 2020) and the community is still smaller. In [CCG+17] Cohn Gordon *et al.* introduced the notion of Asynchronous Ratcheted Trees (ART). These ART are Diffie Hellman based binary trees in which the update process of a node involves entropy coming from both its children. As we detail in section 5.1, MLS is based on a protocol called TreeKEM, which is directly inspired from ART. A main difference is that, in MLS, a single leaf generates, alone, the update data for each of its ancestor nodes while in ART, information from several internal nodes is needed in the update procedure. TreeKEM has been initially formalized in the technical paper [BBR18] and has then evolved to reach the actual description available on the prevailing draft 11 [BBM+20]. Alwen, Coretti, Dodis and Tselekounis formalized in [ACDT20] a Continuous Group Key Agreement (CGKA) derived from the two-party Continuous Key Agreement defined in [ACD19] (*cf.* chapter 3). They provide a security model for CGKA and show that TreeKEM does not achieve optimal FS and PCS security, but prove that using updatable public key encryption can lead to a better security. Our solution is compatible with this improvement. Their analysis only consider a weakly passive adversary, unable to inject packets or drop individual message for instance. They also introduce the notion of Post Compromise Forward Security (PCFS) as the (strictly stronger) combination of FS and PCS holding simultaneously; a current epoch remains secure despite both past and future compromises. In [ACC+19] (to be published in the proceedings of S&P 21), Alwen *et al.* propose an alternative version, called Tainted TreeKEM that achieves a weak version of PCFS against an adaptive and strongly passive adversary, that can learn the users randomness (but can not control it) and decide of its next action depending on its view of the execution so far. Alwen, Coretti Jost and Mularczyk made a step forward to reach active security in [ACJM20]. Their model defines an optimal security notion that TreeKEM does not reach. Three of the above authors refined in [AJM20] the active security notion by defining the insider security. As an active adversary, the insider can control randomness of the parties and controls the network (package order and delivery). He can also "interact with the PKI on behalf of the corrupt users." The authors propose an improved TreeKEM that reaches their insider security, notably by introducing message authentication of the transcript and signature of the packets to be delivered. Finally, Brzuska *et al.* provide in [BCK21] an analyse of the current draft 11, considering both TreeKEM and the Key Schedule on top of it. They analyse the pseudorandomness of the key distribution obtained through those two components. While their model largely differs from the strong model of [AJM20], the author claim that the results are comparable. On a parallel subject, Chase *et al.* recently studied in [CPZ20], the question of the

[1] The last draft has expired the 25th of June. A copy of this last prevailing draft is still accessible at `https://www.ietf.org/archive/id/draft-ietf-mls-protocol-11.txt` and on the affiliated github project `https://github.com/mlswg/mls-protocol`.

privacy of membership in the Signal Group enrolment. Their solution is based on credentials and requires a specific verifiable encryption scheme (*cf.* section 2.7).

5.1.2 The protocol description

Currently in MLS, the authors require an hybrid public key encryption (HPKE) scheme, as designed in [BBLW20] (which was recently studied in [ABH+21]), composed of a KEM to transmit a symmetric key k and an AEAD encryption scheme that encrypts the data under k, as well as a key derivation function. The security of this scheme is examined in [ABH+20]. In the rest of this work, we denote by $\mathsf{Enc}_{pk}(m : r)$ the HPKE encryption of a message m under the public key pk using randomness r. The asymmetric part of Enc is based on an elliptic curve E defined on a finite field $\mathbb{Z}_p$ with base point P of order a prime q. MLS also supposes the existence of a broadcast channel for each group, which distributes the messages to each group member, conserving the order.

TreeKEM. MLS key exchange TreeKEM is based on a binary tree structure (Figure 5.1) where users correspond to leaves and each node is associated to a secret value. Each user U has a long term identity signing key and an initial key package for the encryption scheme Enc (both certified by a PKI). We will simply represent the key package as a public/private key pair (pk_U, sk_U).

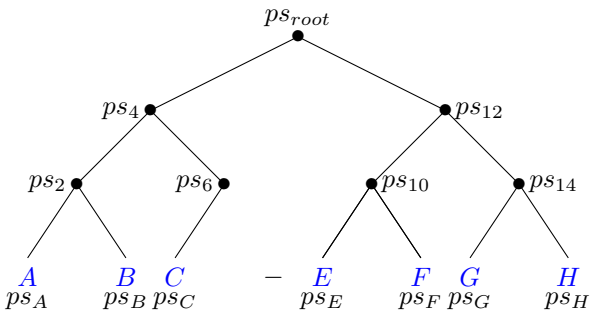


Figure 5.1 – A view of the MLS tree. Nodes are implicitly numbered from left to right, independently from their height. Leaves are associated to a user represented as a letter. Each node i has a secret ps_i. A leaf secret is indexed with its user name.

The group key is derived from the root secret. Each child node knows the secret of each of its ancestors and only of its ancestors. To each node i corresponds a path secret ps_i and a secret and public key $sk_i, pk_i = \mathsf{deriveKeyPair}(ps_i)$ (in the original protocol the keys are derived from an intermediate node secret ns_i itself derived from ps_i. We present a lighter version for the simplicity of the exposure but the complete version is compatible with our solution.) A user knows the secrets ps_i and sk_i in his direct path, composed of himself and his direct ancestors. Moreover, each user keeps an up-to-date global view of the tree, as a hash value of each node's public information.

About the key derivation. In [BBM+20], several suitable cipher suites are described. We focus on one of them for a practical example, for a 128-bit security level. This suite uses X25519 for ECDH computation and SHA256 as a hash function (and base function for HKDF implementation). Following [DJB], the private key sk is obtained from a 256-bit string of secure random data $(sk[0], sk[1], \ldots, sk[255])$ by applying the following transform: $sk[0] = sk[0]\mathsf{AND}248$, $sk[31] = sk[31]\mathsf{AND}127$ then $sk[31] = sk[31]\mathsf{OR}64$. One obtains, when interpreted as an integer

value in little endian, a scalar of the form $2^{254} + 8 \cdot \ell, \ell \in [0, 2^{251} - 1]$. We design by deriveSK the application of SHA256 followed by the above transformation such that for any 32-byte sequence of random data X, deriveSK(X) is a valid secret key for X25519. This encoding can be integrated in the circuit computing the last derivation. The public key is obtained by multiplying the secret key by the base point of the curve. We define, independently from the curve targeted, $(sk_i, pk_i) = \mathsf{deriveKeyPair}(ps_i) = (\mathsf{deriveSK}(ps_i), \mathsf{deriveSK}(ps_i)P)$ where deriveSK is a PRF.

Updates. The path secrets and derived keys are regularly updated. Each update gives birth to a new epoch. To update the tree, a user B generates a new secret ps'_B. The path secrets in the direct path will be successively derived from ps'_B.

We note $\mathsf{H}_p(ps_i)$ for the function $\mathsf{HKDF\text{-}expand}(ps_i, \text{“}path\text{”}, \text{“”}, Hash.length)$. The update mechanism is given in Figure 5.2.

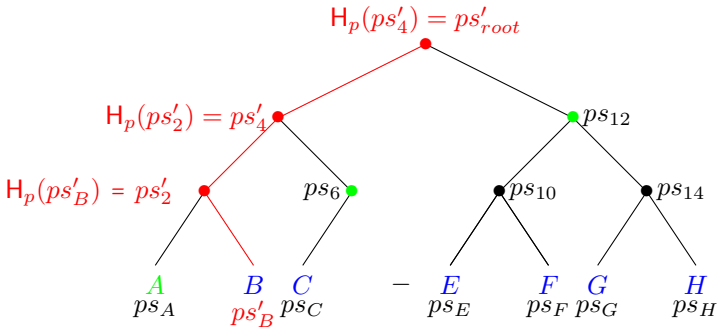


Figure 5.2 – The update process in MLS. User B updates his secrets. Path secrets are updated along his direct path (in red). The update secrets are sent to his copath nodes (in green).

When B updates its secret $ps_B \rightarrow ps'_B$, he first computes his new key package $(sk_B, pk_B) = \mathsf{deriveKeyPair}(ps'_B)$ then he derives the new node data for each node on his path:

- $ps'_2 = \mathsf{H}_p(ps'_B), pk_2 = \mathsf{deriveSK}(ps'_2) \cdot P$;
- $ps'_4 = \mathsf{H}_p(ps'_2), pk_4 = \mathsf{deriveSK}(ps'4) \cdot P$;
- $ps'_{root} = \mathsf{H}_p(ps'_4), pk_{root} = \mathsf{deriveSK}(ps'_{root}) \cdot P$.

Then he sends for each node on his copath the necessary secret material for the users under this node to perform the same update. Following our example in Figure 5.2, B has to send ps'_2 to A, ps'_4 to nodes C and 6 and ps'_{root} to nodes $E, 10, F, 12, G, 14, H$. As a child knows the secret key sk_i for each of its ancestors, B will only have to encrypt ps'_2 under pk_A, ps'_4 under pk_6 and ps'_{root} under pk_{12}. From ps'_2 (respectively ps'_4), A (resp. C) shall be able to compute the root secret.

A late verification. From the root secret is derived an epoch secret S_{E+1}[2]. Before sending his update message, B computes S_{E+1} and uses it to produce a confirmation key. This value shall enable A and C to check that they have derived the correct root secret and so, that they received a correct update. Other mechanisms such as the transmission of the updated view of the tree, or

[2]From the root secret are derived several application keys, however we only focus on how the root secret is updated, not on the how it is used, hence we have not detailed those keys.

of intermediate hash values are provided for a user to check that he received a correct update. All those mechanisms enable a verification after receiving the update information. From then, two different policies are possible: either the update is accepted by all only once each user has confirmed that he received a valid update, this can imply a huge latency, if some users are seldom online, and non valid updates can easily lead to a denial of service. Or the update is validated without such a feedback. In this case, the users that received non valid secret values are ejected from the group *de facto*. In both case, this seriously hampers with the security of the service provided by the protocol.

5.2 Securing MLS updates

We now explain how to combine a ZK protocol (*cf.* section 2.6) and a verifiable encryption scheme (*cf.* section 2.7) to secure the update process in MLS. We first focus on a single step of the update process (a user updates his direct parent) and then explain how this solution can be extended to the global tree.

5.2.1 Server-checking in MLS.

As described in Figure 5.2, lets assume that B generates a new secret ps'_B. Suppose that he computes the required data and, additionally, for each secret, a Pedersen commitment (*cf.* subsection 2.6.4) as follows:

- $\mathsf{deriveKeyPair}(ps'_B)$ to obtain a new key package and $C_B = \mathsf{Com}(ps'_B, r'_B)$;
- $ps'_2 = \mathsf{H}_p(ps'_B)$ the new secret for node 2 and commits to it: $C_2 = \mathsf{Com}(\mathsf{H}_p(ps'_B), r_2)$;
- $(sk'_2, pk'_2) = \mathsf{deriveKeyPair}(ps'_2)$ the new keys for node 2 and the corresponding $C_{sk'_2} = \mathsf{Com}(\mathsf{deriveSK}(\mathsf{H}_p(ps'_B)), r_{sk'_2})$.

Suppose there exists a ZK protocol which, for a PRF f, given public values C_x and C_y, provides a proof that the prover knows opening values x, r_x, r_y such that C_x opens to x with randomness r_x and that C_y opens to $f(x)$ with randomness r_y, which can be written as: $\mathsf{PK}\{x, r_x, r_y : C_x = \mathsf{Com}(x, r_x) \wedge C_y = \mathsf{Com}(f(x), r_y)\}$. Then B can send to the server the public values C_B, C_2, $C_{sk'_2}$, pk'_2 together with a proof $\Pi_2 = \mathsf{PK}\{ps'_B, r_{B'}, r_2, r_{sk'_2} : C_B = \mathsf{Com}(ps'_B, r_{B'}) \wedge C_2 = \mathsf{Com}(\mathsf{H}_p(ps'_B), r_2) \wedge C_{sk'_2} = \mathsf{Com}(\mathsf{deriveSK}(\mathsf{H}_p(ps'_B)), r_{sk'_2}) \wedge pk'_2 = \mathsf{deriveSK}(ps'_2)P\}$ (the last part of the proof being a classic discrete log proof).

On another side, a verifiable encryption scheme VerifEnc such as the Camenish-Damgård described in section 2.7, allows to link the message encrypted with VerifEnc with the data committed in C_2. To sum up, B will send for a node update, the public values C_B, C_2, $C_{sk'_2}$, and pk'_2, the proof Π_2 together with $\mathsf{VerifEnc}_{\mathsf{Enc},pk_A}(ps'_2)$. If the server accepts the proof, then he transmits the public key pk'_2 as well as $\mathsf{VerifEnc}_{\mathsf{Enc},pk_A}(ps'_2)$ to A.

To extend the proof to the complete tree, one has to repeat the above steps for each level. To certify the update value ps'_4 corresponding to the parent node 4, B will send the server values C_4, $C_{sk'_4}$, pk'_4, the proof $\Pi_4 = PK\{ps'_2, r_2, r_4, r_{sk'_4} : C_2 = \mathsf{Com}((ps'_2), r_2) \wedge C_4 = \mathsf{Com}(\mathsf{H}_p(ps'_2), r_4) \wedge C_{sk_4} = \mathsf{Com}(\mathsf{deriveSK}(\mathsf{H}_p(ps'_2)), r_{sk'_4}) \wedge pk'_4 = \mathsf{deriveSK}(\mathsf{H}_p(ps'_2))P\}$ together with $\mathsf{VerifEnc}_{\mathsf{Enc},pk_6}(ps'_4)$. The crucial point is that, as the commitment C_2 is linked with ps'_B in Π_2, it can be used as a base value for Π_4 and so on. Some special care must be taken as we commit, in a group of order q prime, to an element $sk \in \{0,1\}^{256}$ that does not lie naturally in $\mathbb{Z}_q$. We explain how to handle with this in section 5.3.5.

About the server. Several reasons appear for calling on a third party. Firstly, this central node with the largest computational power is the one that can discard invalid updates with the most efficiency. If one relies on users to check for the validity of the data they received, this means that one must wait for each user to process the update and to send back an acknowledgement. As a user can be off-line for a long time, this can be very inefficient. Another solution would be to allow users to adopt the update as soon as they are individually convinced it is correct, while providing a "backup solution". This would probably imply keeping old keys and drastically impoverish FS.

Secondly, in MLS architecture, all the update encrypted messages are gathered and sent as one big message to all the users. It may be of interest to think of a solution where only the needed encryption is sent to a specific user. In this case, only the server will see all the messages together. He is then the only one able to perform a verification on a global proof to see whether all the updates are correctly generated from a single secret seed.

5.3 Zero Knowledge for a PRF on committed input and output

Our solution to secure the updates in MLS requires a ZK protocol which, given public values C_x and C_y, provides the following proof: $\mathsf{PK}\{x, r_x, r_y : C_x = \mathsf{Com}(x, r_x) \wedge C_y = \mathsf{Com}(f(x), r_y)\}$ for any PRF f. This proof mixes algebraic statements (the commitment, which we suppose is a Perdersen Commitment) and a function evaluation. Expressing the algebraic part as a circuit would considerably increase the circuit size and reduce the efficiency. The other way around, one could express each gate of a circuit as an algebraic relation that can be proven with a Σ-protocol, but this solution is clearly non desirable as circuits for hashing may have thousands of gates. Considering this, combining efficiently algebraic and non algebraic proofs has revealed to be an important challenge.

5.3.1 State-of-the-art

In [CGM16], Chase *et al.* propose two constructions, based on Garbled Circuits (*cf.* section 2.6.3), to provide a circuit proof on a committed input. Their first proposal consists, in a first step, in using the values selected by the prover as the garbled input of the circuit to produce a bit-wise commitment to the secret input of the circuit, and, in a second step, proving that this corresponds to a bit-wise commitment to the secret input in the algebraic proof. Their second solution avoids the bitwise commitment by including a one time mac $ax + b$ in the circuit to be garbled. They include this computation aside the original circuit so as to obtain a combination of an algebraic and a non algebraic proof on a single witness ("I know "x" such that $f_1(x) = y$ and $z = g^x$ where f_1 is represented as a circuit). They use their MAC to bind the value committed to in an algebraic commitment and the input to the garbled circuit. In our application, we need to bind the ouput of the circuit to the commitment on the input. In addition, these proposals heavily rely on the garbling and can not be transposed to the non interactive setting.

More recently, Agrawal *et al.* in [AGM18] propose a solution for modular composition of algebraic and non algebraic proofs. Their solution is non interactive, based on Sigma protocols and QAP-based SNARKs (*cf.* section 2.6.3). As explained in their work, the “key ingredient [they] need from a SNARK construction is that the proof contains a multi-exponentiation of the input/output”, thus can not be adapted to the more recent STARKs for instance (see section 2.6.3). They compose it with a proof that the exponents in a multi-exponentiation correspond to values committed to in a collection of commitments. From this result, they show how to obtain proofs for AND, OR and composition of two statements, either algebraic or circuit. The small proofs and the light verification step of

SNARKS are desirable for privacy-preserving credentials or crypto-currencies proofs of solvency. But the prover's high computational effort is not adapted to our application where the verifier turns out to have a larger computation power than the prover.

Finally, Backes *et al.* propose in [BHH+19] an extended version of ZKBoo++ (*cf.* section 2.6.3) that allows algebraic commitments on the secret input of the circuit. Their protocol is non interactive and the computational cost is balanced between the prover and the verifier. Their solution requires to commit to each bit of the secret input and to commit to internal values of the ZKBoo++ circuit proof. Our first contribution is an extension of their work to committed outputs.

5.3.2 An overview of our protocols

As a contribution, we provide three protocols to prove the knowledge of an input x and randoms r_x, r_y, such that, for a public values C_x, C_y, and a function f evaluated as a circuit, $C_x = \mathsf{Com}(x, r_x)$ and $C_y = \mathsf{Com}(f(x), r_y)$.

Our first solution, `ComInOutZK` (Committed Input and Output ZK) is directly inspired from the work of Backes *et al.* ([BHH+19]), which provides a proof of a circuit evaluation on a committed input and public output. We extend their work to a committed output.

Our second contribution comprises two close alternative proposals, `CopraZK` (Commitment and PRF alternative ZK), and `CopraZK+`, which are specific to the case of f being a PRF. The secret x is the PRF key and we evaluate f on a public message m. In `CopraZK`, we consider the circuit that evaluates two equations on x and on another secret input a. We call the results of these equations tag values. The first tag t_1 only depends on $f(x, m)$ and $f(a, m)$. The second tag t_2 depends on x and a. Both t_1 and t_2 also include a value α that is related to the commitments. The idea is that the tags computation binds the values committed to the values used in the circuit. The second alternative, `CopraZK+`, seeks for better efficiency. We consider the circuit that computes a single tag value: $f(x, m) + \alpha x$ (where x is the key of the PRF and α is determined by the public commitment). It shows better efficiency but relies on more theoretical PRF properties. Both `CopraZK` and `CopraZK+` call for some PRF properties, that we discuss in section 5.3.4, and for the homomorphic properties of the commitment.

We compare in Table 5.1 our three solutions with the SNARK based solution of [AGM18]. `CopraZK+` (respectively `CopraZK`) adds a negligible number of algebraic operations. The prover performs 4 (respc. 20) multiplications on the curve (public key operations) and 8 computations in $\mathbb{Z}_q$ (symmetric operations). For the verifier, 6 (respect. 12) computations on the curve and 2 in $\mathbb{Z}_q$ are needed. However, in `CopraZK`, the circuit part of the proof is more than doubled to compute the two tags. Considering ZKBoo, the prover effort is $\mathcal{O}(\sigma|F|)$ symmetric operations, where $|F|$ is the number of AND gates of the circuit and σ the number of rounds. `CopraZK` requires $\mathcal{O}(\sigma(2|F| + |mod|)) + 8$ symmetric operations and 20 public key operations, where $|mod|$ is the size of the circuit for a modular addition which is negligible compared to $|F|$. `CopraZK+` requires $\mathcal{O}(\sigma(|F| + |mod|)) + 8$ symmetric operations and 4 public key operations. For both protocols, the computational cost is dominated by the symmetric part. The size of the proof and the work on the verifier's side are also dominated by the circuit part. One inconvenient is that the security proofs requires non usual hypothesis on the function f. However, the benefice of `CopraZK` is that we provide a reduction to an already known notion for one hypothesis. Another advantage of both `CopraZK` and `CopraZK+` is that they are independant of the circuit based proof chosen. We describe our solutions with ZKBoo as it is an intuitive MPC solution but they can benefit from more recent improvements, as we detail in subsection 5.4.1.

On the opposite side, `ComInOutZK` is valid for any circuit, only requires equivocality of the

	Non interactive	No CRS	Prover's work	Verifier's work	Proof size
SNARK based [AGM18]	yes	no	$\mathcal{O}((\|F\| + \lambda) \cdot pub)$	$\mathcal{O}((\|x\| + \|y\| + \lambda) \cdot pub)$	λ
`CopraZK`	yes	yes	$\mathcal{O}(2\|F\|\lambda \cdot sym)$	$\mathcal{O}(2\|F\|\lambda \cdot sym)$	$\mathcal{O}(2\|F\|\lambda)$
`CopraZK+`	yes	yes	$\mathcal{O}(\|F\|\lambda \cdot sym)$	$\mathcal{O}(\|F\|\lambda \cdot sym)$	$\mathcal{O}(\|F\|\lambda)$
`ComInOutZK`	yes	yes	$\mathcal{O}(\|F\|\lambda \cdot sym + (\|x\|+\|y\|+\lambda) \cdot pub)$	$\mathcal{O}(\|F\|\lambda \cdot sym + (\|x\|+\|y\|+\lambda) \cdot pub)$	$\mathcal{O}((\|F\| + \|x\| + \|y\| + \lambda)\lambda)$

Table 5.1 – Efficiency of the different solutions for a circuit proof on committed input and output. pub stands for the cost of a public key operation (multiplication on the curve), while sym stands for the cost of a symmetric operation. $|F|$ is the circuit size (in terms of multiplication gates), $|x|$ the input size and $|y|$ the output size. In most applications, $|F| >> (|x|, |y|, \lambda)$.

commitment scheme, which is a common hypothesis, and leaves the circuit evaluation untouched. But it requires a non negligible number of algebraic commitments. Considering $|x|$ (respectively $|y|$) the bit size of the input (of the output), we obtain on the prover side $\mathcal{O}(|x| + |y| + 2\sigma)$ public key operations and $\mathcal{O}(\sigma|F|)$ symmetric operations. The verifier's work is equivalent. The proof size of ZKBoo is augmented with $\mathcal{O}(|x| + |y| + 6\sigma)$ curve points which is asymptotically $\mathcal{O}(|x| + |y| + \lambda)$ as σ augments with λ.

On the challenge size When we expose our solutions, in both case we mention a unique challenge, that is used for the algebraic Σ-protocol and for the ZKBoo proof. This means that the challenge space size for the Σ-protocol is 3 and that we shall perform $\lambda/(\log_2(3) - 1)$ rounds to obtain a soundness error in $2^{-\lambda}$. The Σ-protocol can benefit from a larger challenge space, that allows for a single round. As explained in [BHH+19], it is possible to define distinct challenges $e_\rho \in \{1, 2, 3\}$ for each ZKBoo round and a global challenge $e = \sum_{i=1}^{\sigma} 3^i e_i$ for the algebraic Σ-protocols, hence the algebraic part of the proof can be performed a single time.

5.3.3 `ComInOutZK`: a bit wise solution

In [BHH+19], the authors propose a non interactive proof $\mathsf{PK}\{x : C_x = Com(x, r_x) \wedge y = f(x)\}$ based on bit commitments and ZKBoo++. Their optimized solution increases the ZKBoo++ prover's and verifier's work with $\mathcal{O}(|x| + \sigma)$ exponentiations and multiplications on the group $\mathbb{G}$ of order q chosen for the commitment, where $|x|$ is the number of bits of the input x and σ is the number of rounds in ZKBoo++. The proof size grows by $\mathcal{O}(|x| + \sigma)$ group elements and $\mathcal{O}(|x| + \sigma)$ elements in $\mathbb{Z}_q$. We adapt this strategy to the case of a committed output. As the output of the circuit, y, shall remain secret, we will not be able to call ZKBoo++ as a full black box. This is of prime importance when we prove the zero-knowledge property.

The work of Backes *et al.* and our extension rely on a result given by the homomorphic property of a commitment scheme such as Pedersen scheme. For any scalar k, and any two commitments

$\mathsf{Com}(x, r_x)$, $\mathsf{Com}(y, r_y)$, any $k \in \mathbb{Z}_q$, we have that:

$$k \cdot \mathsf{Com}(x, r_x) + \mathsf{Com}(y, r_y) = \mathsf{Com}(kx + y, kr_x + r_y).$$

For any commitment $C_b = \mathsf{Com}(b, r_b)$ to a secret bit b and any public bit β, one can easily compute the commitment of $b \oplus \beta$ as follows:

$$\text{if } \beta = 0, C_{b\oplus\beta} = C_b,$$

$$\text{if } \beta = 1 \text{ then } C_{b\oplus\beta} = \mathsf{Com}(1, 0) - C_b = \mathsf{Com}(1 - b, -r_b).$$

For any $x = \sum_{i=0}^{|x|-1} 2^i x[i]$, denote $C_{x[i]} = Com(x[i], r_{x[i]})$ a commitment to the i-th bit of x. Then $\sum_{i=0}^{|x|-1} 2^i C_{x[i]}$ is a valid commitment to x with opening randomness $\sum_{i=0}^{|x|-1} 2^i r_{x[i]}$. Finally one can easily compute a commitment to $x \oplus \beta$ for an element β as $C_{x\oplus\beta} = \sum_{i=0}^{|x|-1} 2^i C_{x[i]\oplus\beta[i]}$, with opening randomness $\sum_{i=0}^{|x|-1} 2^i (-1)^{\beta[i]} r_{x[i]}$.

The protocol proceeds in two mains steps. First, the prover commits to the bits of the output y of the circuit, and provides a proof that the corresponding commitments are valid bit commitments. The homomorphic properties of the commitment scheme bind the bitwise commitments to the public commitment of y, C_y. In a second steps, the prover executes a ZKBoo proof on the circuit, but, instead of releasing the output share of the unopened view, he only provides a commitment of this share. He then provides the randomness that binds the commitment of the three shares to the bitwise commitments.

The protocol `ComInOutZK`**.** We describe in Figure 5.3 the protocol on a committed output only (`ComOutZK`), for readability reasons. Combining Backes *et al.* protocol for a committed input and ours for a committed output leads to `ComInOutZK`.

Let f be a function: $\mathbb{Z}_2^\ell \to \mathbb{Z}_2^\ell$, $\mathbb{G}$ be a group of prime order q, such that $2^\ell \leq p$. There is a natural embedding $\mathbb{Z}_2^\ell \hookrightarrow \mathbb{G}$. Let P be a generator for this group and Q an element of $\mathbb{G}$ such that $\log_P(Q)$ is unknown. We consider a hash function $\mathsf{h} : \mathbb{Z}_2^* \to \mathbb{Z}_2^{\ell *}$ and a commitment scheme Com that takes as input values in $\mathbb{Z}_q$. The following theorem states the security of our bitwise solution.

Theorem 5.1. *Given that ZKBoo and the Π_j are Σ-protocols with 3-special soundness and honest verifier Zero Knowledge property, and* Com *is a homomorphic and equivocal commitment scheme, then the protocol described in Figure 5.3 is a Σ-protocol with 3-special soundness and honest verifier property.*

Proof. We study separately the three properties a Σ-protocol should verify.

Correctness. It follows by inspection. Assuming the *Prover* and the *Verifier* execute the protocol as described, the *Verifier* never meets a rejection cause and then always accepts.

3-special soundness. Consider an algorithm $\mathcal{E}xt$ that has access to three distinct accepting executions of the protocol on the same commit phase: (a, e_1, p_1), (a, e_2, p_2), and (a, e_3, p_3), $e_1 \neq e_2 \neq e_3$, for a public statement C_y. We show that $\mathcal{E}xt$ can exhibit a witness (x^*, r^*) such that $C_y = \mathsf{Com}(f(x^*), r^*)$. We can not directly call the Extractor from ZKBoo++ as we do not exactly execute ZKBoo. In our protocol, the *Verifier* does not have access to the output of the circuit. However, we show that this difference does not prevent $\mathcal{E}xt$ from succeeding. We describe in the following how $\mathcal{E}xt$ works.
Firstly, from the distinct transcripts, $\mathcal{E}xt$ can obtain three pairs of shares x_{e_1}, x_{e_1+1}, x_{e_2}, x_{e_2+1}, x_{e_3}, x_{e_3+1} (note that, as we detail the extractor on a single round, we do not mention the upper

The *Prover* knows x, $y = f(x)$, and r_y such that $C_y = \mathsf{Com}(f(x), r_y)$. The *Verifier* knows the statement C_y.

Prover

Commit phase

1. samples random $r_{y[j]}$ and commits to the bits of y : $C_{y[j]} = \mathsf{Com}(y[j], r_{y[j]})$ for $j \in [0, |y|]$.
2. computes the commit phase a_{Π_j} for the proofs $\Pi_j = PK\{y[j], r_{y[j]} : C_{y[j]} = \mathsf{Com}(y[j], r_{y[j]}) \wedge y[j] \in \{0, 1\}\}$ for $j \in [0, |y|]$.

for $\rho \in [1, \sigma]$:

3. samples random seeds $k_1^\rho, k_2^\rho, k_3^\rho$.
4. generates the shares $x_1^\rho, x_2^\rho, x_3^\rho = \mathsf{Share}(x, k_1^\rho, k_2^\rho)$ such that $x = x_1^\rho \oplus x_2^\rho \oplus x_3^\rho$.
5. simulates the MPC to obtain three views $w_1^\rho, w_2^\rho, w_3^\rho$.
6. evaluates $y_i^\rho = \mathsf{Output}(w_i^\rho)$, $i \in \{1, 2, 3\}$.
7. commits to the views : $c_i^\rho = \mathsf{h}(w_i^\rho, k_i^\rho)$, $i \in \{1, 2, 3\}$.
8. samples random $r_{y_i^\rho}$ and commits to the outputs : $C_{y_i^\rho} = \mathsf{Com}(y_i^\rho, r_{y_i^\rho})$, $i \in \{1, 2, 3\}$.

$a = ((C_{y_1^\rho}, C_{y_2^\rho}, C_{y_3^\rho}, c_1^\rho, c_2^\rho, c_3^\rho)_\sigma, (C_{y[j]})_{|y|}, (a_{\Pi_j})_{|y|})$

Challenge : $e = \mathsf{h}(a)$

Response phase

1. computes the responses z_{Π_j} for the proofs Π_j

for $\rho \in [1, \sigma]$:

2. $b^\rho = (C_{y_{e+2}^\rho}, c_{e+2}^\rho)$
3. $z^\rho = (w_{e+1}^\rho, k_e^\rho, k_{e+1}^\rho, r_{y_e^\rho}, r_{y_{e+1}^\rho})$
4. $\beta^\rho = y_e^\rho \oplus y_{e+1}^\rho$
5. $C_z^\rho = \sum_{i=0}^{|y|-1} 2^i C_{y[i] \oplus \beta^\rho[i]}$
6. $r_z^\rho = r_{y_{e+2}^\rho} - \sum_{i=0}^{|y|-1} 2^i (-1)^{\beta[i]} r_{y[i]}$

return $p = (e, (b^\rho, z^\rho, r_z^\rho)_\rho), (z_{\Pi_j})_j$

. . . . . .

Verifier(a, p)

1. Parses p as e, $(b^\rho, z^\rho, r_z^\rho)_\sigma$
2. Parses a as $(C_{y_1^\rho}, C_{y_2^\rho}, C_{y_3^\rho}, c_1^\rho, c_2^\rho, c_3^\rho)_\sigma, (C_{y[j]})_{|}y|, a_{\Pi_j})$
3. Reconstructs the proof Π_j (computes a_Π from $(z_{\Pi_j})_j$)
4. Rejects if $C_y \neq \sum_{i=0}^{|y|-1} 2^i C_{y[i]}$

for $\rho \in [1, \sigma]$:

5. runs the MPC protocol to reconstruct w_e^ρ from $w_{e+1}^\rho, k_e^\rho, k_{e+1}^\rho$
6. obtains $y_e^\rho = \mathsf{Output}(w_e)$, $y_{e+1}^\rho = \mathsf{Output}(w_{e+1})$
7. Computes $\beta^\rho = y_e \oplus y_{e+1}$
8. Computes $C_z^\rho = \sum_{i=0}^{|y|-1} 2^i C_{y[i] \oplus \beta^\rho[i]}$
9. Rejects if $C_{y_{e+2}^\rho} \neq \mathsf{Com}(0, r_z) + C_z^\rho$

Reconstructs a and reject if $e \neq h(a)$

Figure 5.3 – The `ComOutZK` protocol. The reconstruct step of the verification consists in computing the commitment a from the response data and check its validity with the challenge. Only the values in a that can not be reconstructed need being sent.

case ρ index). He also gets three pairs of output values y_{e_1}, y_{e_1+1}, y_{e_2}, y_{e_2+1}, y_{e_3}, y_{e_3+1} and the corresponding randomness $r_{y_{e_1}}$, $r_{y_{e_1+1}}$, $r_{y_{e_2}}$, $r_{y_{e_2}+1}$, $r_{y_{e_3}}$, $r_{y_{e_3}+1}$. From the common commitment a, $\mathcal{E}xt$ gets $C_{y_1}, C_{y_2}, C_{y_3}$. As the three transcripts are accepting, $\mathcal{E}xt$ knows that (considering, w.l.o.g., $e_1 = 1, e_2 = 2, e_3 = 3$):

$$C_{y_1} = \mathsf{Com}(y_{e_1}, r_{y_{e_1}}) = \mathsf{Com}(y_{e_3+1}, r_{y_{e_3}+1}).$$

$$C_{y_2} = \mathsf{Com}(y_{e_2}, r_{y_{e_2}}) = \mathsf{Com}(y_{e_1+1}, r_{y_{e_1}+1}).$$

$$C_{y_3} = \mathsf{Com}(y_{e_3}, r_{y_{e_3}}) = \mathsf{Com}(y_{e_2+1}, r_{y_{e_2}+1}).$$

If one of this equality verifies with different opening values, then, due to the equivocability of the commitment scheme, $\mathcal{E}xt$ can extract the trapdoor. Given this knowledge, he can consider any value $\tilde{x}$, compute $\tilde{y} = f(\tilde{x})$ and compute the appropriate randomness to open C_y to $\tilde{y}$.
Now we consider the case when the equalities on the commitments traduce equalities of the openings. $\mathcal{E}xt$ thus obtains three values $y_1 = y_{e_1} = y_{e_3+1}$, $y_2 = y_{e_2} = y_{e_1+1}$, $y_3 = y_{e_3} = y_{e_2+1}$ and a single $y^* = y_1 \oplus y_2 \oplus y_3$. From then, one can call the original ZKBoo extractor that executes back the MPC protocol and obtain three shares $x_1 = x_{e_1} = x_{e_3+1}$, $x_2 = x_{e_2} = x_{e_1+1}$, $x_3 = x_{e_3} = x_{e_2+1}$ and a single $x^* = x_1 \oplus x_2 \oplus x_3$ such that $y^* = f(x^*)$.

Now $\mathcal{E}xt$ needs to extract a randomness r^* that opens C_y to y^*. Using as a subroutine the extractors for the proofs Π_j, $\mathcal{E}xt$ obtains couples $(y'[j], r_{y'[j]})$ for $j \in [0, |y| - 1]$. From the protocol, as the transcripts are accepting ones, $\mathcal{E}xt$ knows that $C_y = \sum_{i=0}^{|y'|-1} 2^i \mathsf{Com}(y'[i], r_{y'[i]})$.
$\mathcal{E}xt$ selects one transcript, for instance e_1.
He computes:

$$\beta = y_{e_1} \oplus y_{e_1+1} \text{ and}$$

$$C_z = \sum_{i=0}^{|y'|-1} 2^i C_{y'[i] \oplus \beta[i]} = \sum_{i=0}^{|y'|-1} 2^i \mathsf{Com}(y'[i] \oplus \beta[i], (-1)^{\beta[i]} r_{y'[i]}).$$

By the protocol, $C_z = C_{y_{e_1+2}} - \mathsf{Com}(0, r_z)$.
If $\sum_{i=0}^{|y'|-1} 2^i (y'[i] \oplus \beta[i]) \neq y_{e_1+2}$ and/or $\sum_{i=0}^{|y'|-1} 2^i (-1)^{\beta[i]} r_{y'[i]} \neq r_{y_{e_1}+2} - r_z$, then again, $\mathcal{E}xt$ obtains the trapdoor of the commitment scheme and can open C_y to the value he wishes.
Otherwise $\sum_{i=0}^{|y'|-1} 2^i (y'[i] \oplus \beta[i]) = y_{e_1+2}$ and $\sum_{i=0}^{|y'|-1} 2^i (y'[i]) = y_{e_1+2} \oplus \beta = y_{e_1+2} \oplus y_{e_1} \oplus y_{e_1+1} = f(x^*)$.
Finally, $\sum_{i=0}^{|y'|-1} 2^i r_{y'[i]}$ opens C_y to $f(x^*)$ and the extractor is done. The running time of the extractor is bounded by the time of running back the MPC protocol (as for the ZKBoo extractor) + the running time of the extractors $\mathcal{E}xt_{\Pi_j}$ + computing one XOR and one commitment. Considering that an extractor for ZKBoo and the extractor for the proofs Π_j run in polynomial time, $\mathcal{E}xt$ also runs in polynomial time.

Zero-knowledge. We consider a simulator $\mathcal{S}im$ that, on input a public statement C_y, shall produce a transcript (a, e, p). As for the soundness, we cannot call directly the ZKBoo simulator, $\mathcal{S}im_{ZKB}$, as the output of the circuit is not part of the statement. $\mathcal{S}im$ runs as follows: he sets e and he samples random tapes k_e, k_{e+1} and random input shares x_e, x_{e+1}. Then he runs the protocol as normal except that, when he meets a binary multiplication gate in the circuit, he cannot compute the real value of the view w_{e+1} (because it would depend on the third view that he cannot compute because he does not know x) so he samples it at random. This is indistinguishable form the real execution as binary multiplication gates are, in a correct execution, randomized with an element from k_{e+2}

that the *Verifier* cannot compute. $\mathcal{S}im$ obtains output values y_e, y_{e+1}. He samples random r_e, r_{e+1}, computes $C_{y_e} = \mathsf{Com}(y_e, r_e)$ and $C_{y_{e+1}} = \mathsf{Com}(y_{e+1}, r_{e+1})$.

In a second step, he samples random $|y| - 1$ bit values $y[j]$, $j \in [1, |y| - 1]$ and associated randomness $r_y[j]$ and computes $C_{y[j]} = \mathsf{Com}(y[j], r[j])$. He executes the proofs Π_j with challenge e. Then he evaluates $C_{y[0]} = C_y - \sum_{j=1}^{|y|-1} C_{y[j]}$. Using the simulator for the proof Π_0, $\mathcal{S}im$ obtains a transcript for Π_0 for a challenge e'. If $e' \neq e$ he runs $\mathcal{S}im_{\Pi_0}$ again. Given that Π_0 is honest verifier, there is a non negligible probability that $e' = e$ within a polynomial time. Defining $\beta = y_e \oplus y_{e+1}$, $\mathcal{S}im$ can compute $C_{y[i]\oplus\beta[i]}$ only from the knowledge of $C_{y[i]}$ and $\beta[i]$. Now $\mathcal{S}im$ samples $r_z \in \mathbb{Z}_p$ and computes $C_{y_{e+2}} = \sum_{j=0}^{|y|-1} C_{y[i]\oplus\beta[i]} + \mathsf{Com}(0, r_z)$. He now has all the elements to produce an accepting transcript.

The transcript of the ZKBoo part of the proof is indistinguishable form a real execution. The elements that $\mathcal{S}im$ produces itself are commitments that will not be opened, hence, by the hiding property of the commitment, the complete simulated transcript is indistinguishable form a real execution of the protocol.

□

5.3.4 CopraZK: a tag-based solution

Let f be a function: $\mathbb{Z}_2^\ell \times \mathbb{Z}_2^* \to \mathbb{Z}_2^\ell$ and m a public input, $m \in \mathbb{Z}_2^*$. Let $\mathbb{G}$ be a group of prime order q, such that $2^\ell \leq q$. There is a natural embedding $\mathbb{Z}_2^\ell \hookrightarrow \mathbb{G}$. Let P be a generator for this group and Q an element of $\mathbb{G}$ such that $\log_P(Q)$ is unknown. Let h be a hash function $\mathbb{Z}_2^* \to \mathbb{Z}_2^{\ell *}$. Let Com be the Pedersen commitment scheme. Those elements are the public parameters of the prover and the verifier. We use a ZKBoo proof for the circuit part, but the protocol and its proof are valid for any circuit based ZK proof or argument. Let C_x, C_y be public commitments, known to the verifier. Let a be a random mask for the secret x. The prover also provides commitments C_a and C_b for values a and $f(a, m)$ and seal all the commitment values by computing $\alpha = \mathsf{h}(C_x||C_y||C_a||C_b)$. The main idea is to consider the circuit that computes two tags $t_1 = f(x, m) + \alpha f(a, m)$ and $t_2 = x + \alpha a$ where a is considered as a second secret entry of the circuit. A MPC in the head proof on this circuit ensures that t_1 and t_2 are correctly computed from two secret values x and a known to the prover. The prover also provides commitments C_a and C_b for values a and $f(a, m)$. Considering Pedersen commitments, we complete the circuit proof with an algebraic proof that the committed values in C_a, C_x, C_y, C_b verify the relations t_1 and t_2. These linear relations together with the properties of f defined below, bind the values of C_x and C_y such that the verifier can be convinced that the value committed in C_y is equal to the evaluation of f on the value committed in C_x. The detailed description of the protocol CopraZK is given in Figure 5.4.

Some specific PRF properties.

As we use Fiat-Shamir to get an non interactive protocol, our proof is settled in the ROM (*cf.* subsection 2.6.2), which would satisfy our hypothesis. However, it seems contradictory to idealize as a random oracle the PRF f that is concretely described as a circuit in the ZKBoo part of the protocol. Hence, the ROM hypothesis only applies to the hash function h that generates the challenge. The 1-varRKA-wPRF and 1-varCI-ow properties defined below provide a way to formalize a security proof when only some properties of the random oracle are needed.

Variation of Related Key Attack properties. Our first notion, 1-varRKA-wPRF is related to the related key attack security recalled in section 2.5. It stipulates that the function f does not

Let f be a function: $\mathbb{Z}_2^\ell \times \mathbb{Z}_2^* \to \mathbb{Z}_2^\ell$ and m a public input, $m \in \mathbb{Z}_2^*$. Let C_y, C_x be public commitments. The *Prover* wants to convince the *Verifier* that he knows x, r_x, r_y such that $C_x = \mathsf{Com}(x, r_x)$ and $C_y = \mathsf{Com}(f(x,m), r_y)$.

Prover

Commit phase:

1. samples $a, r_a, r_b \leftarrow_\$ \mathbb{Z}_2^\ell$.
2. computes $C_a = \mathsf{Com}(a : r_a)$, $C_b = \mathsf{Com}(f(a,m) : r_b)$.
3. computes $\alpha = \mathsf{h}(C_x || C_y || C_a || C_b)$.
4. computes $(t_1, t_2) = (f(x,m) + \alpha f(a,m) \mod q, x + \alpha a \mod q)$.
5. Evaluates the commit phase output a_Π for the Σ protocol $\Pi = PK\{x, r_x, y, r_y, a, r_a, b, r_b : C_x = xP + r_x Q \wedge C_y = yP + r_y Q \wedge C_a = aP + r_a Q \wedge C_b = bP + r_b Q \wedge t_1 = y + \alpha b \wedge t_2 = x + \alpha a\}$

for $\rho \in [1, \sigma]$ (ZKBoo part):

6. samples random tapes $k_1^\rho, k_2^\rho, k_3^\rho$.
7. generates the shares $x_1^\rho, x_2^\rho, x_3^\rho = \mathsf{Share}(x, k_1^\rho, k_2^\rho)$ such that $x = x_1^\rho \oplus x_2^\rho \oplus x_3^\rho$.
8. evaluates the MPC protocol on the circuit Circ that, on entrance values entry (x, a), evaluates $(t_1, t_2) = (f(x,m) + \alpha f(a,m) \mod q, x + \alpha a \mod q)$ and obtains three views $w_1^\rho, w_2^\rho, w_3^\rho$.
9. obtains the output shares : $o_1^\rho = (t_{1,1}, t_{2,1}), o_2^\rho = (t_{1,2}, t_{2,2}), o_3^\rho = (t_{1,3}, t_{2,3})$ such that $t_1 = t_{1,1}^\rho \oplus t_{1,2}^\rho \oplus t_{1,3}^\rho$ and $t_2 = t_{2,1}^\rho \oplus t_{2,2}^\rho \oplus t_{2,3}^\rho$.
10. commits to the views : $c_1^\rho = \mathsf{h}(w_1^\rho, k_1^\rho), c_2^\rho = \mathsf{h}(w_2^\rho, k_2^\rho), c_3^\rho = \mathsf{h}(w_3^\rho, k_3^\rho)$.

$a = C_a, C_b, C_x, t_1, t_2, (c_1^\rho, c_2^\rho, c_3^\rho, o^\rho = (o_1^\rho, o_2^\rho, o_3^\rho))_\rho, a_\Pi$.

Challenge: e = h(a)

Response phase:

1. computes the response z_Π for the proof Π

for $\rho \in [1, \sigma]$:

2. $b^\rho = (o_{e+2}^\rho = (t_{1,e+2}, t_{2,e+2}), c_{e+2}^\rho)$
3. $z^\rho = (w_{e+1}^\rho, k_e^\rho, k_{e+1}^\rho)$

return $p = (e, (b^\rho, z^\rho)_\rho, z_\Pi)$

. . . . . .

Verifier(a, p)

1. Parse p as e, $(b^\rho, z^\rho)_\rho, z_\Pi$
2. Parse a as $C_a, C_b, C_x, t_1, t_2, (c_1^\rho, c_2^\rho, c_3^\rho, o^\rho = (o_1^\rho, o_2^\rho, o_3^\rho))_\rho, a_\Pi$
3. Computes α
4. Reconstruct the proof Π

for $\rho \in [1, \sigma]$ (Verifying the ZKBoo proof):

5. runs the MPC protocol to reconstruct w_e^ρ from $w_{e+1}^\rho, k_e^\rho, k_{e+1}^\rho$
6. obtains $t_{1,e}^\rho, t_{2,e}^\rho = \mathsf{Output}(w_e)$, $t_{1,e+1}^\rho, t_{2,e+1}^\rho = \mathsf{Output}(w_{e+1})$
7. computes $t_{1,e+2}^\rho = t_1 \oplus t_{1,e}^\rho \oplus t_{1,e+1}^\rho$, $t_{2,e+2}^\rho = t_2 \oplus t_{2,e}^\rho \oplus t_{2,e+1}^\rho$

Reconstruct a and reject if $e \neq h(a)$

Figure 5.4 – The CopraZK protocol. The reconstruction means the verification by reconstruction of the challenge in the Fiat-Shamir version.

$\mathsf{Exp}^{\text{1-varRKA-wPRF}}_{\mathcal{A},f,p}(\lambda)$

Let p be a prime, let f be a public efficiently computable function $\mathcal{K} \times \mathcal{D} \to \mathcal{R}$ such that there exists an embedding $\mathcal{R} \hookrightarrow \mathbb{Z}_p$[a]. Let $\mathcal{A}$ be a PPT adversary.

- SetUp. The Challenger samples a key $k \leftarrow_\$ \mathcal{K}$, a random input $r \leftarrow_\$ \mathcal{D}$ and $b \leftarrow_\$ \{0,1\}$. He sends r to $\mathcal{A}$.
- Queries Computation. For $i \in [1,q]$, the adversary computes his queries $q_i = (\alpha_i, t_{1,i}) \in \mathcal{R} \times \mathcal{K}$. He sends those queries to the Challenger.
- Answers. The Challenger samples $g \leftarrow_\$ \mathsf{FF}(\mathcal{K},\mathcal{D},\mathcal{R})$. For $i \in [1,q]$ the Challenger answers to the query q_i with a value $t_{1,i} = f(k,r) - \alpha_i f(\frac{t_{2,i}-k}{\alpha_i}, r)$ if $b=0$, $t_{1,i} = g(k,r) - \alpha_i g(\frac{t_{2,i}-k}{\alpha_i}, r)$ if $b=1$.
- Guess. $\mathcal{A}$ outputs a guess bit $\hat{b}$. The Challenger accepts if $\hat{b} = b$.

[a] the dependency to the security parameter is hidden in the keyspace choice.

Figure 5.5 – The 1-varRKA-wPRF security experiment.

leak information when evaluated as $t_1 = f(x) - \alpha f(\frac{t_2 - x}{\alpha})$. The formal experiment is described in Figure 5.5. The reduction to RKA security tells that if f does not leak any information when evaluated on values $f(\frac{t_2-x}{\alpha})$, then it does not leak more information when evaluated as t_1. This will be of prime importance in the proof of the Zero-Knowledge property of the Sigma protocol `CopraZK`, as the Simulator will not be able to compute t_1 and so will sample it at random.

Definition 5.1 (1-varRKA-wPRF security)**.** *Let $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be an efficiently computable function. f is* 1-varRKA-wPRF*-secure if, for all PPT adversary $\mathcal{A}$ making at most q queries, the quantity $\mathsf{Adv}^{\text{1-varRKA-wPRF}}_{\mathcal{A},f,p}(\lambda)$, defined as:*

$$\left|\Pr\left[\mathsf{Exp}^{\text{1-varRKA-wPRF}}_{\mathcal{A},f,p}(\lambda) = 1 | b = 0\right] - \Pr\left[\mathsf{Exp}^{\text{1-varRKA-wPRF}}_{\mathcal{A},f,p}(\lambda) = 1 | b = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

is negligible.

Proposition 5.2. *Let $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be an efficiently computable function. If f is* 1-$\mathcal{C}$-aRKA-wPRF *secure, then f is* 1-varRKA-wPRF*-secure.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that breaks the 1-varRKA-wPRF security with q queries, in time t for a function $f : \mathcal{K} \times \mathcal{R} \to \mathcal{D}$. Then we build an adversary $\mathcal{B}$ that breaks 1-$\mathcal{C}$-aRKA-wPRF security of f with $q+1$ queries and runs the same time as $\mathcal{A}$. The adversary $\mathcal{B}$ uses $\mathcal{A}$ as follows:

- SetUp. The Challenger for 1-$\mathcal{C}$-aRKA-wPRF security samples a bit b, a key $k \leftarrow_\$ \mathcal{K}$ and a random function $g \leftarrow_\$ \mathsf{Func}(\mathcal{K} \times \mathcal{D}, \mathcal{R})$. He samples a random input r that he sends to $\mathcal{B}$. $\mathcal{B}$ sends r to $\mathcal{A}$.
- First Query. $\mathcal{B}$ queries his oracle with the identity polynomial. He receives a value y.
- q queries. For $i \in [1,q]$ adversary $\mathcal{A}$ sends its challenger (played by $\mathcal{B}$) a query $\{t_i, \alpha_i\}_{i \in [1,q]}$. $\mathcal{B}$ defines the circuit $C_i : X \to \frac{X - t_i}{\alpha_i}$. He queries his own oracle with C_i. $\mathcal{B}$ receives value y_i. He sends $z_i = y - \alpha_i y_i$ to $\mathcal{A}$.
- Guess. After his q queries, $\mathcal{A}$ answers with a bit b'. $\mathcal{B}$ transmits b' to its own Challenger.

$\mathcal{B}$ perfectly simulates the oracle for $\mathcal{A}$. The probability that $\mathcal{B}$ wins is exactly the probability that $\mathcal{A}$ wins. □

$\mathsf{Exp}^{\text{1-varCI-ow}}_{\mathcal{A},\mathsf{H},p}$

Let p be a prime number, f be an efficiently computable function $\mathcal{K} \times \mathcal{D} \to \mathcal{R}$, such that there exists an embedding $\mathcal{R} \hookrightarrow \mathbb{Z}_p$ and $\mathcal{A}$ a PPT adversary. The experiment rus as follows:

- SetUp. The Challenger selects a random key k and samples a random public input r. He sends r to $\mathcal{A}$.
- Queries. For $i \in [1, q]$ the adversary computes his queries $q_i = (\alpha_i, t_{1,i}) \in \mathcal{K} \times \mathcal{K}$. He sends them to the Challenger. He receives a value $t_{2,i} = f(k, r) - \alpha_i f(\frac{k - t_{1,i}}{\alpha_i}, r) \in \mathbb{R}$.
- Invert. $\mathcal{A}$ sends a couple (x, j) with $x \in \mathcal{K}$ and j the index of a query. The Challenger accepts if $f(x, r) - \alpha_j f(\frac{x - t_{1,j}}{\alpha_j}, r) = t_{2,j}$.

Figure 5.6 – The 1-varCI-ow security experiment.

Note that, in this definition as well as in the description of CopraZK, the input r is sampled at random. However in MLS, the input in the HKDF successive node secrets derivations is always equal to 0. This can be fixed by changing the value of this input from zero to the hash value of some public data depending on the updated tree, that are already computed in the protocol, such as the tree view for instance.

Variation of correlated input one-wayness. Our second notion, 1-varCI-ow, formalizes the fact that the equation $eq_1 : t_1 = f(x) - \alpha f(\frac{t_2 - x}{\alpha})$ is hard to solve. This will be of prime importance in the soundness proof. The corresponding security experiment is given in Figure 5.6. The definition is related to Correlated Input one-wayness, as defined in section 2.5. However, contrary to 1-varRKA-wPRF, there is no simple reduction between the two notions.

Definition 5.2 (1-varCI-ow security)**.** *Let $f : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be an efficiently computable function. f is said to be* 1-varCI-ow*-secure if, for all PPT adversary $\mathcal{A}$ making q queries, the following advantage: $\mathsf{Adv}^{\text{1-varCI-ow}}_{\mathcal{A},f,p}(\lambda) = \Pr\left[\mathsf{Exp}^{\text{1-varCI-ow}}_{\mathcal{A},f,p}(\lambda) = 1\right]$ is negligible.*

In real life, most of the widely used hash functions are build from compression functions, based on a keyed block cipher (SHA-2 family) or on an unkeyed block defined permutation (Keccak family). PRF are mostly derived from those kind of hash functions. This the case of the HKDF-expand function, built from HMAC and instantiated in MLS with a SHA family. If few literature exists concerning RKA or CI one-wayness for real-life PRF, we state that finding a preimage or correlations on inputs and outputs of a hash function is among the most difficult problem for symmetric cryptography experts.

CopraZK security

The following theorem states the security of CopraZK.

Theorem 5.3. *Let f be a* 1-varRKA-wPRF *and* 1-varCI-ow *secure function,* H *be a secure hash function and* Com *a homomorphic commitment scheme. Then the* CopraZK *protocol described in Figure 5.4 defines a Σ-protocol with 3-special soundness and computational honest verifier zero-knowledge.*

Proof. Correctness. If the prover and the verifier act as required by the protocol, then the verifier always accept for a valid proof. In fact, the verifier knows C_x and C_y and so computes the same value

for α. Then, considering the proof of knowledge of a linear relation between two Pedersen Commitments Π as detailed in subsection 2.6.4, from z_Π, C_x and C_y, the verifier can reconstruct the correct a_Π. From $(b_\rho, z_\rho)_{\rho\in[1,\sigma]}$ the verifier can reconstruct the values $(c_1^\rho, c_2^\rho, c_3^\rho, o^\rho = (o_1^\rho, o_2^\rho, o_3^\rho))_{\rho\in[1,\sigma]}$ and t_1, t_2 (and check that the values he gets is equal to the t_1, t_2 that were transmitted in a that he used to reconstruct Π). From these values, he can compute the correct challenge value e and accept.

3-special soundness. Suppose we are given three distinct transcripts with a common commitment phase (a, e_1, p_1), (a, e_2, p_2) and (a, e_3, p_3). From the 2-special soundness of the Sigma protocol Π, one gets values $\tilde{x}, \tilde{y}, \tilde{a}, \tilde{b}\tilde{r_x}, \tilde{r_y}, \tilde{r_a}$, and $\tilde{r_b}$ such that $C_x = \tilde{x}P + \tilde{r_x}Q$, $C_y = \tilde{y}P + \tilde{r_y}Q$, $C_a = \tilde{a}P + \tilde{r_a}Q$, $C_b = \tilde{b}P + \tilde{r_b}Q$ and $t_1 = \tilde{y} - \alpha\tilde{b}$ and $t_2 = \tilde{x} - \alpha\tilde{a}$. From the 3-special soundness of ZKBoo, one can extract x', a' such that $t_1 = f(x', m) + \alpha f(a', m)$ and $t_2 = x' - \alpha a'$, where t_1, t_2 are fixed values, the same as the one for the proof Π. Now this gives valid extraction the following equalities verify: $x' = \tilde{x}$, $a' = \tilde{a}$, $y' = \mathsf{PRF}(\tilde{x})$, $b' = f(a')$. The extractor can return the triple $x', r_{x'}, r_{y'}$. Hence we look at the probability that one or more of the previous equations do not verify. We call **Bad** this event, and separate it in four subcases.

Case $\mathbf{Bad_1}$. Only one equality among $\tilde{y} = f(x')$, $\tilde{a} = a'$, $\tilde{b} = f(a')$, $\tilde{x} = x'$ does not verify. As the relations t_1, t_2 verify, this can never happen, $\Pr[\mathbf{Bad_1} = 0]$.

Case $\mathbf{Bad_2}$. Two equalities out of the four do not verify. Due to relations t_1, t_2 we have :

1. if $\tilde{x} = x'$ and $\tilde{y} = f(x')$ then $\tilde{a} = a'$ and $\tilde{b} = f(a')$;
2. if $\tilde{x} = x'$ and $\tilde{b} = f(a')$ then $\tilde{a} = a'$ and $\tilde{y} = f(x')$;
3. if $\tilde{a} = a'$ and $\tilde{y} = f(x')$ then $\tilde{b} = f(a')$ and $\tilde{x} = x'$;
4. if $\tilde{a} = a'$ and $\tilde{b} = f(a')$ then $\tilde{y} = f(x')$ and $\tilde{x} = x'$;

If only $\tilde{x} = x'$ and $\tilde{a} = a'$, we look for the probability that $\tilde{y} \neq f(x')$ and $\tilde{b} \neq f(a')$. It is obvious that if one inequality stands, so does the other. Calling the binding security of Pedersen commitment and the one-wayness of h, α is a fixed value and so are $\tilde{x}$, $\tilde{a}$, y', b' and t_1 and t_2. Then if $\tilde{y} \neq f(x')$ and $\tilde{b} \neq f(a')$, it must be that x' is a solution to the equation $f(x) - \alpha f(\frac{x-t_2}{\alpha}) = t_1$, breaking the 1-varCI-ow security of f.

Now if only $\tilde{y} = f(x')$ and $\tilde{b} = f(a')$, again, x' is a solution to the equation $f(x) - \alpha f(\frac{x-t_2}{\alpha}) = t_1$, breaking the 1-varCI-ow security of f. So $\Pr[\mathbf{Bad_2}] \leq 2\,(\epsilon_{ow} + \epsilon_{\mathsf{Com}} + \epsilon_{\text{1-varCI-ow}})$, where ϵ_{ow} (respectively $\epsilon_{\mathsf{Com}}, \epsilon_{\text{1-varCI-ow}}$) denote the advantage terms for the one-way security of h (respectively the binding property of the commitment and the 1-varCI-ow security of f).

Case $\mathbf{Bad_3}$. Only one equality verifies. Due to relations t_1 and t_2, this case is impossible and $\Pr[\mathbf{Bad_3}] = 0$.

Case $\mathbf{Bad_4}$. Finally suppose no equation verifies. Again, calling the binding property of the Pedersen commitment and the one wayness of h, α is a fixed value. As in case $\mathbf{Bad_2}$, if the proof accepts then it must be that x' is a solution to $f(x) - \alpha f(x - t_2/\alpha) = t_1$. This is again the 1-varCI-ow security of f and $\Pr[\mathbf{Bad_4}] \leq \epsilon_{\text{1-varCI-ow}}$. Finally, given the one wayness of h, the binding property of the commitment, and the 1-varCI-ow-security of f, the triple $(x', r_{x'}, r_{y'})$ is a valid witness.

Zero-Knowledge We describe a simulator that outputs in polynomial time a transcript such that the distribution of its transcript is indistinguishable from the distribution of the real execution transcripts. The simulator $\mathcal{S}im$ works as follows: firstly, it samples a random $x \in \mathbb{Z}_2^\ell$. Then the Simulator acts as a real *Prover*, following steps 1, 2 and 3 : he samples random a, b, r_a, r_b and computes

the commitments C_a, C_b. Then he computes α. Given his random x, a, b and the related α, he can compute $t_2 = x - \alpha a$ and samples t_1 at random. This is possible because of the 1-varRKA-wPRF security of f. We note here that the relation between the real value corresponding to the public commitment C_x, C_y, a, b and t_1, t_2 are not verified *a priori*. As seen in Section 2.6.4, this shall not disturb the Simulation. $\mathcal{S}im$ calls the Simulator of the algebraic proof Π, $\mathcal{S}im_\Pi$ on $t_1, t_2, C_x, C_a, C_y, C_b$ and obtains a transcript (a_Π, e_Π, z_Π) Given the output of the circuit (t_1, t_2), $\mathcal{S}im$ calls the ZKBoo Simulator, $\mathcal{S}im_{ZKBoo}$, on e to obtain a transcript $(a_{ZKB} = c_1, c_2, c_3, (o_1, o_2, o3), e, (b, z))$ In particular, $o_1 \oplus o_3 \oplus o_3 = (t_1, t_2)$ and the view and random seeds in z enable to reconstruct the ZKBoo proof. Finally the Simulator returns $(C_a, C_b, t_1, t_2, a_{ZKB}, a_\Pi), e, (b, z, z_\Pi))$.

The transcript obtained from $\mathcal{S}im_\Pi$ and $\mathcal{S}im_{ZKBoo}$ are (statistically) indistinguishable from real transcripts. If f is 1-varRKA-wPRF-secure, then sampling a random t_1 is indistinguishable from the real distribution of t_1 and finally, the output distribution of $\mathcal{S}im$ is indistinguishable from the real execution output. □

5.3.5 CopraZK+: toward a more efficient version of CopraZK

As previously, let f be a function: $\mathbb{Z}_2^\ell \times \mathbb{Z}_2^* \to \mathbb{Z}_2^\ell$ and m a public input, $m \in \mathbb{Z}_2^*$. Let $\mathbb{G}$ be a group of prime order q, such that $2^\ell \leq q$. Let P be a generator for this group and Q an element of $\mathbb{G}$ such that $\log_P(Q)$ is unknown. Let h be a hash function $\mathbb{Z}_2^* \to \mathbb{Z}_2^{\ell *}$ and Com be the Pedersen commitment scheme. They are the public parameters of the protocol.

Let C_x, C_y be public commitments, known to the verifier. The idea to get a more efficient protocol is to consider the circuit that only computes the tag $t = f(x, m) + \alpha x$ where α is a public coefficient derived from the commitment values C_x and C_y. A MPC in the head proof on this circuit ensures that t is correctly computed from a secret value x known to the prover. Considering Pedersen commitments, we complete the circuit proof with an algebraic proof that the committed values in C_x and C_y verify the relation t. This linear relation plus the properties of f defined below, bind the values of C_x and C_y such that the verifier can be convinced that the value committed in C_y is equal to the evaluation of f on the value committed in C_x. A complete description is given in Figure 5.7. We depict our protocol using ZKBoo for the circuit part, but the proof adapts to any circuit based ZK proof.

Yet another PRF property.

As for our protocol CopraZK, we avoid considering the PRF f as a random oracle. Hence, this also require to call some more subtil PRF properties. Firstly, we need its dual function $\tilde{f}$ to be correlation intractable (*cf.* subsection 2.5.2) with respect to the family of relations $\mathcal{R}_{a,b} : \{x, y : y = ax + b\}$ for a, b random values.

Secondly, we need to be sure that the tag does not leak information on the key. We define a general linear input deviation resistant PRF (glider-PRF) as follows:

Definition 5.3 (glider-PRF security)**.** *A function family $f \in \mathsf{FF}(\mathcal{K}, \mathcal{D}, \mathcal{R})$ (with appropriate domain and range) is said to be a* glider-PRF *if for all PPT adversary $\mathcal{A}$, and a random $\alpha \leftarrow\!\!\$\, \mathbb{R}$, there exists a negligible function* negl *such that:*

$$\Pr\left[x \leftarrow \mathcal{A}(\alpha)^{\mathcal{O}_g} : g \leftarrow\!\!\$\, \mathsf{F}(\mathcal{D}, \mathcal{R})\right] - \Pr\left[x \leftarrow \mathcal{A}(\alpha)^{\mathcal{O}_{f(k,\cdot)+\alpha\cdot}} : k \leftarrow\!\!\$\, \mathcal{K}\right] \leq \mathsf{negl}(\lambda)\,.$$

As for 1-varCI-ow, we do not have a reduction to a more classical (or at least already defined) property. We consider as an important, yet challenging, research target to explore and formalise

Let C_y, C_x be public commitments and m a public message. The *Prover* wants to convince the *Verifier* that he knows x, r_x, r_y such that $C_x = \mathsf{Com}(x, r_x)$ and $C_y = \mathsf{Com}(f(x, m), r_y)$.

Prover

Commit phase:

1. computes $\alpha = \mathsf{h}(C_x || C_y)$.
2. computes $t = f(x, m) + \alpha x \mod q$.
3. Evaluates the commit phase output a_Π for the Σ protocol $\Pi = PK\{x, r_x, y, r_y : C_x = xP + r_xQ \wedge C_y = yP + r_yQ \wedge t = y + \alpha x\}$

for $\rho \in [1, \sigma]$ (ZKBoo part):

4. samples random tapes $k_1^\rho, k_2^\rho, k_3^\rho$.
5. generates the shares $x_1^\rho, x_2^\rho, x_3^\rho = \mathsf{Share}(x, k_1^\rho, k_2^\rho)$ such that $x = x_1^\rho \oplus x_2^\rho \oplus x_3^\rho$.
6. evaluates the MPC protocol on the circuit Circ that, on entrance values entry (x, a), evaluates $t = f(x, m) + \alpha x \mod q$ and obtains three views $w_1^\rho, w_2^\rho, w_3^\rho$.
7. obtains the output shares : $o_1^\rho = t_1, o_2^\rho = t_2, o_3^\rho = t_3$ such that $t = t_1^\rho \oplus t_2^\rho \oplus t_3^\rho$.
8. commits to the views : $c_1^\rho = \mathsf{h}(w_1^\rho, k_1^\rho), c_2^\rho = \mathsf{h}(w_2^\rho, k_2^\rho), c_3^\rho = \mathsf{h}(w_3^\rho, k_3^\rho)$.

$a = t, (c_1^\rho, c_2^\rho, c_3^\rho, o^\rho = (o_1^\rho, o_2^\rho, o_3^\rho))_\rho, a_\Pi$.

Challenge: e = h(a)

Response phase:

1. computes the response z_Π for the proof Π

for $\rho \in [1, \sigma]$:

2. $b^\rho = (o_{e+2}^\rho = t_{e+2}, c_{e+2}^\rho)$
3. $z^\rho = (w_{e+1}^\rho, k_e^\rho, k_{e+1}^\rho)$

returns $p = (e, (b^\rho, z^\rho)_\rho, z_\Pi)$

. . . . . .

Verifier(a, p)

1. parses p as $e, (b^\rho, z^\rho)_\rho, z_\Pi$
2. parses a as $t, (c_1^\rho, c_2^\rho, c_3^\rho, o^\rho = (o_1^\rho, o_2^\rho, o_3^\rho))_\rho, a_\Pi$
3. computes α
4. reconstruct the proof Π

for $\rho \in [1, \sigma]$ (Verifying the ZKBoo proof):

5. runs the MPC protocol to reconstruct w_e^ρ from $w_{e+1}^\rho, k_e^\rho, k_{e+1}^\rho$
6. obtains $t_e^\rho = \mathsf{Output}(w_e), t_{e+1}^\rho = \mathsf{Output}(w_{e+1})$
7. computes $t_{e+2}^\rho = t \oplus t_e^\rho \oplus t_{e+1}^\rho$.

reconstructs a and rejects if $e \neq h(a)$

Figure 5.7 – Our protocol `CopraZK+`.

the security properties expected and achieved by real life PRF. Nowadays, even the already stated properties such as correlation intractability are only proven (partially) for algebraic constructions that are not efficient and not used in everyday life protocols.

Theorem 5.4 states the security of `CopraZK+`.

Theorem 5.4. *Given that ZKBoo and the Π_j are Σ protocols with 3 (respectively 2) special soundness and honest verifier Zero Knowledge property (in their interactive form), that* Com *is a homomorphic commitment scheme, and that f is a* glider-PRF *function family such that $\tilde{f}$ is correlation intractable*

relatively to relations $\{\mathrm{R}_{a,b} : \{x, y : y = ax + b\}\}$*, then the protocol described in Figure 5.7 is a* Σ*-protocol with 3-special soundness and full Zero-Knowledge.*

Proof. Correctness. If the prover and the verifier act as required by the protocol, then the verifier always accept for a valid proof. In fact, the verifier knows C_x and C_y and so computes the same value for α. Then, considering the proof Π of knowledge of a linear relation between two Pedersen commitments (*cf.* subsection 2.6.4), from z_Π, C_x and C_y, the verifier can reconstruct the correct a_Π. From $(b_\rho, z_\rho)_{\rho\in[1,\sigma]}$ the verifier can reconstruct the values $(c_1^\rho, c_2^\rho, c_3^\rho, o^\rho = (o_1^\rho, o_2^\rho, o_3^\rho))_{\rho\in[1,\sigma]}$ and t (and check that the t he gets is equal to the t that was transmitted in a that he used to reconstruct Π). From these values, he can compute the correct challenge value e and accept.

3-special soundness. Suppose we are given three distinct transcripts with a common commitment phase (a, e_1, p_1), (a, e_2, p_2) and (a, e_3, p_3). From the 2-special soundness of the Sigma protocol Π, one extract $\tilde{x}, \tilde{y}, \tilde{r_x}, \tilde{r_y}$ such that $C_x = \tilde{x}P + \tilde{r_x}Q, C_y = \tilde{y}P + \tilde{r_y}Q$ and $t = \alpha\tilde{x} + \tilde{y}$. From the 3-special soundness of ZKBoo, one extracts x' such that $t = f(x', m) + \alpha x'$, where t is a fixed value, the same as the one for the proof Π. Correlation intractability of $\tilde{f}$ ensures that $x' \neq \tilde{x}$ happens with negligible probability. We note that the correlation intractability of $\tilde{f}$ requires the input of f to be randomized. In MLS, this supposes considering a random value (for instance the hash of the tree view) instead of the constant 0.

Zero Knowledge. We build a simulator $\mathcal{S}im$ as follows: $\mathcal{S}im$ computes t from the public values. The glider-PRF-security of f ensures that t does not give any information on x. Then he calls the Simulator of ZKBoo, $\mathcal{S}im_{ZKBoo}$, as a subroutine and obtains a transcript $(a_{ZKBoo,e,z_{ZKBoo}})$. Then he calls the simulator for the Sigma protocol Π, $\mathcal{S}im_\Pi$, as a second subroutine, on the challenge e and obtains a second transcript (a_Π, e, z_Π) (as $\mathcal{S}im_\Pi$ shall work for any challenge). Finally, as the transcripts given by $\mathcal{S}im_{ZKBoo}$ and $\mathcal{S}im_\Pi$ are indistinguishable from real transcript, the output distribution of $\mathcal{S}im$ is indistinguishable from the real execution output. In the context of MLS, the tag t must be accessible to the server only. A user who would receive its valid update and access the tag could compute the secret of its child, which he should not.

□

Group order and commitments. In our proofs, we consider commitments and discrete logarithm proofs in cyclic groups of order q, and circuits input and output that naturally lie in F_q. This may not be the case. Considering the X25519 key derivation described in section 5.1.2, a new user's secret ps'_B is a random element in $\{0,1\}^{256}$, which, when interpreted as an integer, can be larger than q. As explained in [BHH+19], it is possible to consider $ps'_B \bmod q$ for the commitment and to include to a modular computation in the circuit. If q is close enough to 2^{256} then it is a simple comparison and subtraction. This requires around 2 000 gates, which is negligible compared to our circuit size. Another solution is to directly sample ps'_B in F_q. This can be done by rejection sampling or as follow: sample X sufficiently big compared to $\log_2(q)$ ($\log_2(X) > \log_2(q) + 64$ as advised by the NIST for instance), then simply considering $X \bmod q$ can be done with a negligible bias. For all the intermediate values in the tree, the first method can be applied. The last step is the commitment of the secret key $sk = \mathsf{deriveSK}(X)$. For this element, we directly consider the encoding provided with the curve. The commitment C_{sk} of sk in a group of order q will result in the same implicit reduction modulo q than the computation of the public key. Then we can produce an AND ZK proof that the value committed to in C_{sk} is the discrete log of the pk: $PK\{sk : C_{sk} = skP + rQ \wedge pk = skP\}$.

Prover (ms)		Verifier (ms)	
Generating random	21	Loading file	1
Sharing secrets	1	Generating challenge	0
Running circuit	534	Verifying	799
Committing	20		
Total generating proof data	578	Total verifying	800
Proof size (MB)	3.3		

Table 5.2 – Running meantime of the *Prover* and the *Verifier* over 1 000 executions for 136 rounds.

5.4 Implementation results

We now discuss the efficiency of our protocol `CopraZK`, both in terms of time and size of the proof. We focused on `CopraZK` because it was more prone than `ComInOutZK` to benefit from optimized MPC protocols such as KKW ([KKW18]), that are designed for large circuits. And compared to `CopraZK+`, it can be seen as a worst case, and so we can get an upper bound on the efficiency. Considering the table Table 5.1, we can easily estimate performances of `CopraZK+` as being half the result we present here.

Our implementation. We implemented the circuit part of our `CopraZK` protocol on top of the ZKBoo code, available at `https://github.com/Sobuno/ZKBoo`. ZKBoo has been fairly optimized since then but this code was easily accessible and we considered it as a good starting point. The code provides ZKBoo versions for elementary operations. These functions operate on three views and each binary AND call is randomized as recommended in the ZKBoo description. The code provides ZKBoo versions of operations on 32 bits vectors: addition, XOR, AND, addition with a constant. It also provides a ZKBoo version of a SHA_256 circuit, that comprises around 23300 binary AND gates. We implemented a ZKBoo version for the HMAC function, with two calls to SHA_256, and a 256-bit addition. The function HMAC corresponds to HKDF − expand when the desired output length equals the output length of the underlying hash function. Our final circuit, with input x and a computes $t_1 = x + a$ and $t_2 = \mathsf{HMAC}(x) + \mathsf{HMAC}(a)$ for a total of 93696 AND gates. We did not implement the modular reduction. However, as we expect our entries x and a (similarly $\mathsf{HMAC}(x)$ and $\mathsf{HMAC}(a)$) to be in the cyclic group $\mathbb{Z}_q$ (*cf.* section 5.3.5), t_1 and t_2 may only exceed q by one q. Hence modular reduction can be instantiated as a comparison and subtraction if necessary. Using Cingulata (a compiler toolchain for homomorphic encryption, available at `https://github.com/CEA-LIST/Cingulata`), we estimated the number of AND gates for this operation on 8 and 16 bits integers, and obtained respectively 48 and 103 AND gates. From this result, we can expect that a modular reduction on 256 bits integer can be implemented using around 2000 AND gates. This number being far from representative in our circuit, we did not considered this operation in a basic implementation. We consider the running time for a soundness parameter $\sigma = 80$ (corresponding to a soundness error of 2^{-80}), requiring 136 rounds.

Times are given for four steps on the *Prover* side: the generation of the randomness, the sharing of the secret inputs, the evaluation of the circuit, and the committing phase. One step is representative on the *Verifier* side: checking the consistency of the received proof. Our tests were run on a Dell laptop with Processor IntelCore i7-7600U CPU running a single core at $2,8$ Ghz with 15.5 GB of

RAM. Results are given in Table 5.2. We see that the prover's running time is better than the verifier's one, which is not the case in [GMO16], running ZKBoo on the mere SHA_256 circuit. Running a proof on a bigger circuit increases the verifier's load more than the prover's one. The running time of the prover is around half a second when the verification takes around 0.8 seconds. In [GMO16], the author show that a parallelized implementation can seriously improve those results (with 8 threads, they divide the running time by 3.4 for the prover and by 5 for the verifier). This experimental proof size is large, and we examine in the following section the optimization possibilities.

5.4.1 Improvements

The first improvement, ZKBoo++, is given in [CDG+17]. The authors meticulously analyse which data should be sent by the prover and which one can be directly computed by the verifier. They show that one can cut by more than half the size of the proof, at no computational cost. They implemented their solution on an optimized circuit for SHA_256 with 22272 AND gates and obtained a proof size of 618KB for a soundness error of 2^{-128} (48% of ZKBoo proof size on the same circuit). For a soundness parameter $\sigma = 80$ they obtain 385KB. Considering this reduction, our own proof would drop to 1.6MB. Moreover, the optimized SHA_256 circuit enables to save around 4000 AND gates, cancelling the cost of the modular reduction. We now turn to the KKW protocol [KKW18]. The authors use another MPC solution to decrease the number of rounds required to reach a desired soundness security. They show that the improvement one can expect on the size of the proof depends on the number of AND gates. Considering an average of 95000 AND gates for our circuit, the proof size drops by 70KB compared to ZKBoo++. Hence we could obtain proofs around 90KB.

5.4.2 Comparison with SNARKs solutions

Agrawal *et al.* proposed in [AGM18] a ZK protocol mixing algebraic commitments on input and output and circuit evaluation (comIOSnark). It is worth comparing their solution to ours. No implementation is available in [AGM18], however they estimate the prover's work to four exponentiation in addition to the number of exponentiations for computing the SNARK proof when the $Verifier$ has to perform 4 exponentiations and 30 pairings. To compare to more practical results, we recall the performances of Pinocchio ([PHGR13]), on which the protocol of [AGM18] bases its description. The implementation for Pinocchio is performed on a single core of a 2.67 GHz Intel Core i7 with 8 GB of RAM, which is comparable to our test setting. A proof on a SHA_1 evaluation with 23785 multiplication gates requires, first, a public key generation of 11 seconds. The proof computation takes 15.7 seconds. As expected however, the $Verifier$'s running time is only around 10 ms and the proof size is 288 bytes. Even if optimizations can be performed, the $Prover$'s work is far more important than in our solution. Hence we believe that, depending on the applications targeted, either one or the other solution might be of interest.

6 Conclusion

On our way to design alternative or more secure messaging protocols, we were confronted to several problems, among them: usability, efficiency, compatibility with already deployed applications. For some of them we found acceptable solutions, which we recall in the first part of this conclusion, while other are still open or led to even more questions, that we evoke in the second part.

6.1 Summary of the Results

This thesis focused on end-to-end secure instant messaging, a massively implemented protocol. We presented state of the art of the literature concerning the two users ratcheted key exchange, as we believe that a well understood abstraction of the security goals targeted by a real-life protocol is a necessary first step toward more secure solutions. This state of the art reveals a large panel of possible security degrees, from the basic but already adopted Double Ratchet to the ideal optimal post compromise security for which no efficient concrete solution is available for now.

Given this background, our first contribution was turned to a concrete usability problem: the possible security of a multi device version of secure instant messaging. On the way to the final protocol, we provided a multicast solution, the Ratcheted Dynamic Multicast, that provides forward secrecy and some post-compromise security. This new multicast comes lightly over the existing and already massively adopted Signal, to compose our final construction. As we have underlined in the introduction, a user mainly chooses its messaging application relatively to the number of peers that have made the same choice. Hence we think of interest trying to improve a protocol that has already been chosen by billions of users. It seems more difficult to move all these users to a different, but multi-device born - application. However, implementations results shall be improved and should encourage us, to optimize the actual code to get a more accurate comparison with the already deployed Sesame solution.

Remaining on the path of usability, our second contribution aims at giving cryptographic tools to patch a default in the currently developed (and maybe soon deployed) MLS group messaging solution. We combine powerful cryptographic tools: Zero-knowledge and verifiable encryption, to enforce the security of the final protocol and avoid attacks likely to happen. Our contribution is also more theoretical, as we provide new zero-knowledge protocols that provide proofs on mixed algebraic and circuit based statements.

6.2 Open Problems

Towards a more efficient RDM. The construction we proposed for the ratcheted dynamic multicast is the naive one. It thus requires a non negligible number of asymmetric keys to be generated and the update message is a concatenation of messages encrypted for each key. Even if

the management of many keys as well as basic asymmetric cryptography can be implemented in a transparent way (in fact, Signal does it), one can legitimately wonder whether a more optimized solution can be designed. We feel that this ties up with the updatable public key primitives, that are necessary to obtain optimal PCS in ratcheted key exchange, but are non efficient enough yet.

Preserving deniability. As we have seen in section 3.2.3, the deniability of Signal has only recently been formally studied. An evident follow up would be to study the deniability of our multi-device version of the protocol. More generally, we think that this feature could help convincing users of the benefits from end-to-end security, as it can be illustrated with concrete examples such as protection of the sources for journalists, who are sometimes (illegally) summoned to reveal their informer's identity. Hence it would be reasonable to consider the deniability feature of the different ratcheted key exchange proposition to see if some designs are fundamentally incompatible with this property. It would provide a supplementary parameter to consider when looking for an optimal solution.

Verifiable encryption. As we mentioned in section 2.7, there are interesting ways towards verifiable encryption schemes more efficient than the cut-and-choose solution of Camenish and Damgård. Another way that seems to us of interest was introduced in [CCL+19]. It is based on groups with subgroups of unknown orders and the Castagnos Laguillaumie framework. We refer to [Tuc20] for more information. However this would require a complete re examination of the MLS protocol as we would loose the main benefit of the Camenish-Damgård solution, which is the possibility to keep the encryption scheme required in MLS specification.

On PRF properties. The more general open problem that appears to us concerns the study of real-life PRF properties. As explained in subsection 2.5.1, the random oracle model is not necessarily a desirable long term solution. It is more to be considered as an acceptable compromise while no better result is available. However, we note that there is a gap between the known security properties of real life hash functions (as for instance the SHA families) and the security properties one requires in cryptographic proofs. Hence it appears as an important challenge to reconcile theory and practice. We are conscious that this problem has been opened for some decades now.

Bibliography

[16] *The Zcash Ceremony*. 2016. Link. (Cit. on p. 48).

[21] *Trillian IM*. 2021. Link. (Cit. on p. 59).

[ABD+21] Erdem Alkim, Joppe W. Bos, Léo Ducas, Patrick Longa, Ilya Mironov, Michael Naehrig, Valeira Nikolaenko, Chris Peiker, Ananth Raghunathan, and Dougla Stebila. *FrodoKEM Learning With Errors Key Encapsulation. NIST PQ Competition Round 3 specification*. specification. June 2021. Link. (Cit. on p. 79).

[ABH+20] Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. *Analysing the HPKE Standard*. Cryptology ePrint Archive, Report 2020/1499. https://eprint.iacr.org/2020/1499. 2020 (cit. on p. 127).

[ABH+21] Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. *Analysing the HPKE Standard*. In: *Advances in Cryptology – EUROCRYPT 2021*. Ed. by Anne Canteaut and François-Xavier Standaert. Cham: Springer International Publishing, 2021, pp. 87–116 (cit. on p. 127).

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. *The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES*. In: *CT-RSA 2001*. Ed. by David Naccache. Vol. 2020. LNCS. Springer, Heidelberg, Apr. 2001, pp. 143–158 (cit. on pp. 26, 30).

[ACC+19] Joël Alwen, Margarita Capretto, Miguel Cueto, Chethan Kamath, Karen Klein, Ilia Markov, Guillermo Pascual-Perez, Krzysztof Pietrzak, Michael Walter, and Michelle Yeo. *Keep the Dirt: Tainted TreeKEM, Adaptively and Actively Secure Continuous Group Key Agreement*. Cryptology ePrint Archive, Report 2019/1489. https://eprint.iacr.org/2019/1489. 2019 (cit. on p. 126).

[ACD18] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. *The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol*. Cryptology ePrint Archive, Report 2018/1037. https://eprint.iacr.org/2018/1037. 2018 (cit. on p. 79).

[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 pp. viii, 7, 71–73, 79, 126).

[ACDT20] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. *Security Analysis and Improvements for the IETF MLS Standard for Group Messaging*. In: *CRYPTO 2020, Part I*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12170. LNCS. Springer, Heidelberg, Aug. 2020, pp. 248–277 (cit. on p. 126).

[ACF20] Gildas Avoine, Sébastien Canard, and Loïc Ferreira. *Symmetric-Key Authenticated Key Exchange (SAKE) with Perfect Forward Secrecy*. In: *CT-RSA 2020*. Ed. by Stanislaw Jarecki. Vol. 12006. LNCS. Springer, Heidelberg, Feb. 2020, pp. 199–224 (cit. on p. 33).

[ACJM20] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. *Continuous Group Key Agreement with Active Security*. In: *TCC 2020, Part II*. Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12551. LNCS. Springer, Heidelberg, Nov. 2020, pp. 261–290 (cit. on p. 126).

[AGM18] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. *Non-Interactive Zero-Knowledge Proofs for Composite Statements*. In: *CRYPTO 2018, Part III*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10993. LNCS. Springer, Heidelberg, Aug. 2018, pp. 643–673 (cit. on pp. 130–132, 145).

[AH16] Erinn Atwater and Urs Hengartner. *Shatter: Using Threshold Cryptography to Protect Single Users with Multiple Devices*. In: *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. WiSec '16. Darmstadt, Germany: Association for Computing Machinery, 2016, pp. 91–102. ISBN: 9781450342704. Link. (Cit. on pp. 84, 87).

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. *Ligero: Lightweight Sublinear Arguments Without a Trusted Setup*. In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 2087–2104 (cit. on p. 48).

[AHM+14] Joël Alwen, Martin Hirt, Ueli Maurer, Arpita Patra, and Pavel Raykov. *Key-Indistinguishable Message Authentication Codes*. In: *SCN 14*. Ed. by Michel Abdalla and Roberto De Prisco. Vol. 8642. LNCS. Springer, Heidelberg, Sept. 2014, pp. 476–493 (cit. on p. 30).

[AJM20] Joël Alwen, Daniel Jost, and Marta Mularczyk. *On The Insider Security of MLS*. Cryptology ePrint Archive, Report 2020/1327. https://eprint.iacr.org/2020/1327. 2020 (cit. on p. 126).

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. "Proof Verification and the Hardness of Approximation Problems". In: *J. ACM* 45.3 (May 1998), pp. 501–555. ISSN: 0004-5411. Link. (Cit. on p. 47).

[AR00] Michel Abdalla and Leonid Reyzin. *A New Forward-Secure Digital Signature Scheme*. In: *ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. LNCS. Springer, Heidelberg, Dec. 2000, pp. 116–129 (cit. on p. 32).

[BBB+19] Olivier Blazy, Angèle Bossuat, Xavier Bultel, Pierre-Alain Fouque, Cristina Onete, and Elena Pagnin. *SAID: Reshaping Signal into an Identity-Based Asynchronous Messaging Protocol with Authenticated Ratcheting*. In: *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 294–309. Link. (Cit. on p. 70).

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *Scalable, transparent, and post-quantum secure computational integrity*. Cryptology ePrint Archive, Report 2018/046. https://eprint.iacr.org/2018/046. 2018 (cit. on p. 48).

[BBKS07] M. Bellare, A. Boldyreva, K. Kurosawa, and J. Staddon. "Multirecipient Encryption Schemes: How to Save on Bandwidth and Computation Without Sacrificing Security". In: *IEEE Transactions on Information Theory* 53.11 (2007), pp. 3927–3943 (cit. on p. 26).

[BBLW20] Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher Wood. *Hybrid Public Key Encryption: draft-barnes-cfrg-hpke-06*. draft-barnes-cfrg-hpke-06. Sept. 2020. Link. (Cit. on p. 127).

[BBM+20] Richard Barnes, Benjamin Beurdouche, Jon Millican, Emad Omara, Katriel Cohn-Gordon, and Raphael Robert. *The Messaging Layer Security (MLS) Protocol: draft-ietf-mls-protocol-11*. draft-ietf-mls-protocol-11. Dec. 2020. Link. (Cit. on pp. x, 8, 126, 127).

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. *Public-Key Encryption in a Multi-user Setting: Security Proofs and Improvements*. In: *EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. LNCS. Springer, Heidelberg, May 2000, pp. 259–274 (cit. on pp. 25, 26, 97).

[BBR18] Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. *TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups*. May 2018. Link. (Cit. on p. 126).

[BC10] Mihir Bellare and David Cash. *Pseudorandom Functions and Permutations Provably Secure against Related-Key Attacks*. In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 666–684 (cit. on p. 43).

[BCC+14] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. *The Hunting of the SNARK*. Cryptology ePrint Archive, Report 2014/580. https://eprint.iacr.org/2014/580. 2014 (cit. on p. 47).

[BCD+16] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. *Frodo: Take off the Ring! Practical, Quantum-Secure Key Exchange from LWE*. In: *ACM CCS 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1006–1018 (cit. on p. 79).

[BCK21] Chris Brzuska, Eric Cornelissen, and Konrad Kohbrok. *Cryptographic Security of the MLS RFC, Draft 11*. Cryptology ePrint Archive, Report 2021/137. https://eprint.iacr.org/2021/137. 2021 (cit. on p. 126).

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. *Keying Hash Functions for Message Authentication*. In: *CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. LNCS. Springer, Heidelberg, Aug. 1996, pp. 1–15 (cit. on p. 29).

[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. 34).

[BCM11] Mihir Bellare, David Cash, and Rachel Miller. *Cryptography Secure against Related-Key Attacks and Tampering*. In: *ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. LNCS. Springer, Heidelberg, Dec. 2011, pp. 486–503 (cit. on p. 43).

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. *Aurora: Transparent Succinct Arguments for R1CS*. In: *EUROCRYPT 2019, Part I*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11476. LNCS. Springer, Heidelberg, May 2019, pp. 103–128 (cit. on p. 48).

[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. 21, 23).

[Bel06] Mihir Bellare. *New Proofs for NMAC and HMAC: Security without Collision-Resistance*. In: *CRYPTO 2006*. Ed. by Cynthia Dwork. Vol. 4117. LNCS. Springer, Heidelberg, Aug. 2006, pp. 602–619 (cit. on p. 30).

[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. 55).

[BF01] Dan Boneh and Matthew K. Franklin. *Identity-Based Encryption from the Weil Pairing*. In: *CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. LNCS. Springer, Heidelberg, Aug. 2001, pp. 213–229 (cit. on p. 70).

[BF07] Manuel Barbosa and Pooya Farshim. *Randomness Reuse: Extensions and Improvements*. In: *11th IMA International Conference on Cryptography and Coding*. Ed. by Steven D. Galbraith. Vol. 4887. LNCS. Springer, Heidelberg, Dec. 2007, pp. 257–276 (cit. on p. 26).

[BFG+20] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. *Towards post-quantum security for Signal's X3DH handshake*. In: *Proc. 27th Conference on Selected Areas in Cryptography (SAC) 2020*. Ed. by Michael J. Jacobson Jr., Orr Dunkelman, and Colin O'Flynn. LNCS. Springer, Oct. 2020 (cit. on p. 64).

[BFGJ17] Jacqueline Brendel, Marc Fischlin, Felix Günther, and Christian Janson. *PRF-ODH: Relations, Instantiations, and Impossibility Results*. In: *CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. LNCS. Springer, Heidelberg, Aug. 2017, pp. 651–681 (cit. on p. 66).

[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. 38, 39).

[BG93] Mihir Bellare and Oded Goldreich. *On Defining Proofs of Knowledge*. In: *CRYPTO'92*. Ed. by Ernest F. Brickell. Vol. 740. LNCS. Springer, Heidelberg, Aug. 1993, pp. 390–420 (cit. on p. 44).

[BGB04] Nikita Borisov, Ian Goldberg, and Eric Brewer. *Off-the-record communication, or, why not to use PGP*. In: Jan. 2004, pp. 77–84 (cit. on p. 59).

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. *Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys*. In: *CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. LNCS. Springer, Heidelberg, Aug. 2005, pp. 258–275 (cit. on p. 27).

[BHH+19] Michael Backes, Lucjan Hanzlik, Amir Herzberg, Aniket Kate, and Ivan Pryvalov. *Efficient Non-Interactive Zero-Knowledge Proofs in Cross-Domains Without Trusted Setup*. In: *PKC 2019, Part I*. Ed. by Dongdai Lin and Kazue Sako. Vol. 11442. LNCS. Springer, Heidelberg, Apr. 2019, pp. 286–313 (cit. on pp. x, 9, 131, 132, 143).

[BJM97] 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. 32, 33).

[BK03] Mihir Bellare and Tadayoshi Kohno. *A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications*. In: *EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. LNCS. Springer, Heidelberg, May 2003, pp. 491–506 (cit. on p. 43).

[BK09] Alex Biryukov and Dmitry Khovratovich. *Related-Key Cryptanalysis of the Full AES-192 and AES-256*. In: *ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. LNCS. Springer, Heidelberg, Dec. 2009, pp. 1–18 (cit. on p. 43).

[BKR00] Mihir Bellare, Joe Kilian, and Phillip Rogaway. "The Security of the Cipher Block Chaining Message Authentication Code". In: *Journal of Computer and System Sciences* 61.3 (2000), pp. 362–399. ISSN: 0022-0000. Link. (Cit. on p. 30).

[BKR94] Mihir Bellare, Joe Kilian, and Phillip Rogaway. *The Security of Cipher Block Chaining*. In: *CRYPTO'94*. Ed. by Yvo Desmedt. Vol. 839. LNCS. Springer, Heidelberg, Aug. 1994, pp. 341–358 (cit. on p. 18).

[BM99a] 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. 32).

[BM99b] Simon Blake-Wilson and Alfred Menezes. *Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol*. In: *PKC'99*. Ed. by Hideki Imai and Yuliang Zheng. Vol. 1560. LNCS. Springer, Heidelberg, Mar. 1999, pp. 154–170 (cit. on p. 32).

[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. 30).

[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 pp. 34, 35).

[BPS00] Olivier Baudron, David Pointcheval, and Jacques Stern. *Extended Notions of Security for Multicast Public Key Cryptosystems*. In: *ICALP 2000*. Ed. by Ugo Montanari, José D. P. Rolim, and Emo Welzl. Vol. 1853. LNCS. Springer, Heidelberg, July 2000, pp. 499–511 (cit. on pp. 25, 97).

[BR93] Mihir Bellare and Phillip Rogaway. *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*. In: *ACM CCS 93*. Ed. by Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby. ACM Press, Nov. 1993, pp. 62–73 (cit. on p. 41).

[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. 32–36, 92).

[BR95] Mihir Bellare and Phillip Rogaway. *Provably Secure Session Key Distribution: The Three Party Case*. In: *27th ACM STOC*. ACM Press, May 1995, pp. 57–66 (cit. on pp. 35, 36).

[BR96] Mihir Bellare and Phillip Rogaway. *The Exact Security of Digital Signatures: How to Sign with RSA and Rabin*. In: *EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. LNCS. Springer, Heidelberg, May 1996, pp. 399–416 (cit. on p. 18).

[BSJ+17] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. *Ratcheted Encryption and Key Exchange: The Security of Messaging*. In: *CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. LNCS. Springer, Heidelberg, Aug. 2017, pp. 619–650 (cit. on pp. viii, 33, 71–74).

[BSSW06] Xavier Boyen, Hovav Shacham, Emily Shen, and Brent Waters. *Forward-secure signatures with untrusted update*. In: *ACM CCS 2006*. Ed. by Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati. ACM Press, Oct. 2006, pp. 191–200 (cit. on p. 32).

[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 p. 32).

[CCD+16] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. *A Formal Security Analysis of the Signal Messaging Protocol*. Cryptology ePrint Archive, Report 2016/1013. https://eprint.iacr.org/2016/1013. 2016 (cit. on p. 66).

[CCD+17] Katriel Cohn-Gordon, Cas J. F. 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 2017, Paris, France, April 26-28, 2017*. 2017, pp. 451–466 (cit. on pp. viii, ix, 8, 61, 63, 64, 66, 67, 72, 89, 107, 113, 117, 119, 120).

[CCG+17] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. *On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees*. Cryptology ePrint Archive, Report 2017/666. https://eprint.iacr.org/2017/666. 2017 (cit. on p. 126).

[CCG+18] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. *On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees*. In: *ACM CCS 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM Press, Oct. 2018, pp. 1802–1819 (cit. on p. 85).

[CCG+19] Katriel Cohn-Gordon, Cas Cremers, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. *Highly Efficient Key Exchange Protocols with Optimal Tightness*. In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 767–797 (cit. on p. 68).

[CCG16] Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. *On Post-compromise Security*. In: *CSF 2016Computer Security Foundations Symposium*. Ed. by Michael Hicks and Boris Köpf. IEEE Computer Society Press, 2016, pp. 164–178 (cit. on pp. 33, 89).

[CCL+19] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. *Two-Party ECDSA from Hash Proof Systems and Efficient Instantiations*. In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 191–221 (cit. on p. 148).

[CCR15] Ran Canetti, Yilei Chen, and Leonid Reyzin. *On the Correlation Intractability of Obfuscated Pseudorandom Functions*. Cryptology ePrint Archive, Report 2015/334. https://eprint.iacr.org/2015/334. 2015 (cit. on p. 42).

[CCR16] Ran Canetti, Yilei Chen, and Leonid Reyzin. *On the Correlation Intractability of Obfuscated Pseudorandom Functions*. In: *TCC 2016-A, Part I*. Ed. by Eyal Kushilevitz and Tal Malkin. Vol. 9562. LNCS. Springer, Heidelberg, Jan. 2016, pp. 389–415 (cit. on p. 42).

[CD00] Jan Camenisch and Ivan Damgård. *Verifiable Encryption, Group Encryption, and Their Applications to Separable Group Signatures and Signature Sharing Schemes*. In: *ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. LNCS. Springer, Heidelberg, Dec. 2000, pp. 331–345 (cit. on pp. 54, 55).

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. *Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives*. In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 1825–1842 (cit. on pp. 51, 145).

[CDGM19] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. *SEEMless: Secure End-to-End Encrypted Messaging with less Trust*. In: *ACM CCS 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 1639–1656 (cit. on p. 70).

[CDV21] Andrea Caforio, F. Betül Durak, and Serge Vaudenay. *Beyond Security and Efficiency: On-Demand Ratcheting with Security Awareness*. In: *PKC 2021, Part II*. Ed. by Juan Garay. Vol. 12711. LNCS. Springer, Heidelberg, May 2021, pp. 649–677 (cit. on pp. 71, 80).

[CF12] Cas J. F. Cremers and Michele Feltz. *Beyond eCK: Perfect Forward Secrecy under Actor Compromise and Ephemeral-Key Reveal*. In: *ESORICS 2012*. Ed. by Sara Foresti, Moti Yung, and Fabio Martinelli. Vol. 7459. LNCS. Springer, Heidelberg, Sept. 2012, pp. 734–751 (cit. on p. 36).

[CFKN20] Cas Cremers, Jaiden Fairoze, Benjamin Kiesl, and Aurora Naska. *Clone Detection in Secure Messaging: Improving Post-Compromise Security in Practice*. In: *ACM CCS 2020*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. ACM Press, Nov. 2020, pp. 1481–1495 (cit. on p. 68).

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. *The Random Oracle Methodology, Revisited (Preliminary Version)*. In: *30th ACM STOC*. ACM Press, May 1998, pp. 209–218 (cit. on pp. 41, 42).

[CGM+11] Sandy Clark, Travis Goodspeed, Perry Metzger, Zachary Wasserman, Kevin Xu, and Matt Blaze. *Why (Special Agent) Johnny (Still) Can't Encrypt: A Security Analysis of the APCO Project 25 Two-Way Radio System*. In: *USENIX Security 2011*. USENIX Association, Aug. 2011 (cit. on p. 5).

[CGM16] Melissa Chase, Chaya Ganesh, and Payman Mohassel. *Efficient Zero-Knowledge Proof of Algebraic and Non-Algebraic Statements with Applications to Privacy Preserving Credentials*. In: *CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 499–530 (cit. on p. 130).

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. *A Forward-Secure Public-Key Encryption Scheme*. In: *EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. LNCS. Springer, Heidelberg, May 2003, pp. 255–271 (cit. on pp. 33, 59, 84).

[CJZ11] John Callas, Alan Johnston, and Philip Zimmermann. *ZRTP: Media Path Key Agreement for Unicast Secure RTP*. IETF web site. Apr. 2011. Link. (Cit. on p. 33).

[CK01a] Ran Canetti and Hugo Krawczyk. *Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels*. Cryptology ePrint Archive, Report 2001/040. https://eprint.iacr.org/2001/040. 2001 (cit. on p. 38).

[CK01b] 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. 34, 35, 38, 72).

[CK02] Ran Canetti and Hugo Krawczyk. *Universally Composable Notions of Key Exchange and Secure Channels*. In: *EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. LNCS. Springer, Heidelberg, Apr. 2002, pp. 337–351 (cit. on p. 38).

[CPZ20] Melissa Chase, Trevor Perrin, and Greg Zaverucha. *The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption*. In: *ACM CCS 2020*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. ACM Press, Nov. 2020, pp. 1445–1459 (cit. on pp. 55, 126).

[CS97] Jan Camenish and Markus Stadler. *Proof Systems for General Statements about Discrete Logarithms*. Technical Report No.260. Dept. of Computer Science, ETH Zurich, 1997. Link. (Cit. on p. 45).

[CS98] Ronald Cramer and Victor Shoup. *A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack*. In: *CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. LNCS. Springer, Heidelberg, Aug. 1998, pp. 13–25 (cit. on pp. 24, 26).

[Dam10] Ivan Damgård. *On Sigma Protocols*. lecture. Aarhus University, 2010. Link. (Cit. on p. 45).

[DDO+16] Alexander De Luca, Sauvik Das, Martin Ortlieb, Iulia Ion, and Ben Laurie. *Expert and Non-Expert Attitudes towards (Secure) Instant Messaging*. In: *Proceedings of the Twelfth USENIX Conference on Usable Privacy and Security*. SOUPS '16. Denver, CO, USA: USENIX Association, 2016, pp. 147–157. ISBN: 9781931971317 (cit. on pp. vi, 6).

[Del07] Cécile Delerablée. *Identity-Based Broadcast Encryption with Constant Size Ciphertexts and Private Keys*. In: *ASIACRYPT 2007*. Ed. by Kaoru Kurosawa. Vol. 4833. LNCS. Springer, Heidelberg, Dec. 2007, pp. 200–215 (cit. on p. 27).

[DF03] Yevgeniy Dodis and Nelly Fazio. *Public Key Trace and Revoke Scheme Secure against Adaptive Chosen Ciphertext Attack*. In: *PKC 2003*. Ed. by Yvo Desmedt. Vol. 2567. LNCS. Springer, Heidelberg, Jan. 2003, pp. 100–115 (cit. on p. 27).

[DFGS20] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. *A Cryptographic Analysis of the TLS 1.3 Handshake Protocol*. Cryptology ePrint Archive, Report 2020/1044. https://eprint.iacr.org/2020/1044. 2020 (cit. on p. 40).

[DGGL21] Antonio Dimeo, Felix Gohla, Daniel Goßen, and Niko Lockenvitz. *SoK: Multi-Device Secure Instant Messaging*. Cryptology ePrint Archive, Report 2021/498. https://eprint.iacr.org/2021/498. 2021 (cit. on p. 84).

[DGK05] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. *Secure Off-the-Record Messaging*. In: *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*. WPES '05. Alexandria, VA, USA: Association for Computing Machinery, 2005, pp. 81–89. Link. (Cit. on p. 60).

[DGK06] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. *Deniable authentication and key exchange*. In: *ACM CCS 2006*. Ed. by Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati. ACM Press, Oct. 2006, pp. 400–409 (cit. on p. 69).

[DJB] D.J.Bernstein. *A State-of-the-art Diffie Hellman Function*. Link. (Cit. on p. 127).

[DNDS19] Sergej Dechand, Alena Naiakshina, Anastasia Danilova, and Matthew Smith. *In Encryption We Don't Trust: The Effect of End-to-End Encryption to the Masses on User Perception*. In: *2019 IEEE European Symposium on Security and Privacy (EuroSP)*. 2019, pp. 401–415 (cit. on pp. v, 5).

[DPP07] Cécile Delerablée, Pascal Paillier, and David Pointcheval. *Fully Collusion Secure Dynamic Broadcast Encryption with Constant-Size Ciphertexts or Decryption Keys*. In: *PAIRING 2007*. Ed. by Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto. Vol. 4575. LNCS. Springer, Heidelberg, July 2007, pp. 39–59 (cit. on pp. 27, 28).

[DV18] F. Betül Durak and Serge Vaudenay. *Bidirectional Asynchronous Ratcheted Key Agreement with Linear Complexity*. Cryptology ePrint Archive, Report 2018/889. https://eprint.iacr.org/2018/889. 2018 (cit. on pp. 71, 80).

[DV19] F. Betül Durak and Serge Vaudenay. *Bidirectional Asynchronous Ratcheted Key Agreement with Linear Complexity*. In: *IWSEC 19*. Ed. by Nuttapong Attrapadung and Takeshi Yagi. Vol. 11689. LNCS. Springer, Heidelberg, Aug. 2019, pp. 343–362 (cit. on pp. viii, 7, 71, 73–75, 77, 79, 80).

[DVW92] Whitfield Diffie, Paul C Van Oorschot, and Michael J. Wiener. *Authentication and authenticated key exchanges*. In: *Proc. Designs, Codes and Cryptography*. Ed. by editor. June 1992, pp. 107–125. Link. (Cit. on p. 33).

[EI04] Institute of Electrical and Electronics Engineers (IEEE). *Standard Specifications for Public Key Cryptography - Amendment 1: Additional Techniques - Std. 1363a*. 2004. Link. (Cit. on p. 30).

[ElG84] Taher ElGamal. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. In: *CRYPTO'84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. LNCS. Springer, Heidelberg, Aug. 1984, pp. 10–18 (cit. on p. 26).

[Fac17] Facebook. *Messenger Secret Conversation, Technical Whitepaper, Version 2.0*. May 2017 (cit. on p. 84).

[FG14] Marc Fischlin and Felix Günther. *Multi-Stage Key Exchange and the Case of Google's QUIC Protocol*. In: *ACM CCS 2014*. Ed. by Gail-Joon Ahn, Moti Yung, and Ninghui Li. ACM Press, Nov. 2014, pp. 1193–1204 (cit. on pp. 39, 69).

[FMB+16] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Jörg Schwenk, and Thorsten Holz. *How Secure is TextSecure?* In: *2016 IEEE European Symposium on Security and Privacy (EuroS P)*. 2016, pp. 457–472 (cit. on pp. 63, 66, 68).

[FN94] Amos Fiat and Moni Naor. *Broadcast Encryption*. In: *CRYPTO'93*. Ed. by Douglas R. Stinson. Vol. 773. LNCS. Springer, Heidelberg, Aug. 1994, pp. 480–491 (cit. on p. 27).

[FOPS04] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. "RSA-OAEP Is Secure under the RSA Assumption". In: *Journal of Cryptology* 17.2 (Mar. 2004), pp. 81–104 (cit. on p. 24).

[FS87] Amos Fiat and Adi Shamir. *How to Prove Yourself: Practical Solutions to Identification and Signature Problems*. In: *CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. LNCS. Springer, Heidelberg, Aug. 1987, pp. 186–194 (cit. on pp. 41, 46).

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. *On the Cryptographic Applications of Random Functions*. In: *CRYPTO'84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. LNCS. Springer, Heidelberg, Aug. 1984, pp. 276–288 (cit. on p. 41).

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. *Quadratic Span Programs and Succinct NIZKs without PCPs*. In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 626–645 (cit. on p. 47).

[GHS+21] Yaron Gvili, Julie Ha, Sarah Scheffler, Mayank Varia, Ziling Yang, and Xinyuan Zhang. *TurboIKOS: Improved Non-interactive Zero Knowledge and Post-quantum Signatures*. In: *Applied Cryptography and Network Security - 19th International Conference, ACNS 2021, Kamakura, Japan, June 21-24, 2021, Proceedings, Part II*. Ed. by Kazue Sako and Nils Ole Tippenhauer. Vol. 12727. Lecture Notes in Computer Science. Springer, 2021, pp. 365–395. Link. (Cit. on p. 51).

[GM05] Simson L. Garfinkel and Robert C. Miller. *Johnny 2: A User Test of Key Continuity Management with S/MIME and Outlook Express*. In: *Proceedings of the 2005 Symposium on Usable Privacy and Security*. SOUPS '05. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2005, pp. 13–24. ISBN: 1595931783. Link. (Cit. on p. 5).

[GM15] Matthew D. Green and Ian Miers. *Forward Secure Asynchronous Messaging from Puncturable Encryption*. In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 305–320 (cit. on pp. 59, 84).

[GM82] Shafi Goldwasser and Silvio Micali. *Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information*. In: *14th ACM STOC*. ACM Press, May 1982, pp. 365–377 (cit. on p. 21).

[GM84] Shafi Goldwasser and Silvio Micali. "Probabilistic encryption". In: *Journal of Computer and System Sciences* 28.2 (1984), pp. 270–299. ISSN: 0022-0000. Link. (Cit. on p. 21).

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. *ZKBoo: Faster Zero-Knowledge for Boolean Circuits*. In: *USENIX Security 2016*. Ed. by Thorsten Holz and Stefan Savage. USENIX Association, Aug. 2016, pp. 1069–1083 (cit. on pp. 49, 145).

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM J. Comput.* 18 (1989), pp. 186–208 (cit. on p. 44).

[GOR11] Vipul Goyal, Adam O'Neill, and Vanishree Rao. *Correlated-Input Secure Hash Functions*. In: *TCC 2011*. Ed. by Yuval Ishai. Vol. 6597. LNCS. Springer, Heidelberg, Mar. 2011, pp. 182–200 (cit. on pp. 42, 43).

[GS02] Craig Gentry and Alice Silverberg. *Hierarchical ID-Based Cryptography*. In: *ASIACRYPT 2002*. Ed. by Yuliang Zheng. Vol. 2501. LNCS. Springer, Heidelberg, Dec. 2002, pp. 548–566 (cit. on p. 75).

[GSV21] Yaron Gvili, Sarah Scheffler, and Mayank Varia. *BooLigero: Improved Sublinear Zero Knowledge Proofs for Boolean Circuits*. Cryptology ePrint Archive, Report 2021/121. https://eprint.iacr.org/2021/121. 2021 (cit. on p. 48).

[HKKP21] Keitaro Hashimoto, Shuichi Katsumata, Kris Kwiatkowski, and Thomas Prest. *An Efficient and Generic Construction for Signal's Handshake (X3DH): Post-Quantum, State Leakage Secure, and Deniable*. In: *Public-Key Cryptography – PKC 2021*. Ed. by Juan A. Garay. Cham: Springer International Publishing, 2021, pp. 410–440. ISBN: 978-3-030-75248-4 (cit. on p. 64).

[HMS03] Dennis Hofheinz, Jörn Müller-Quade, and Rainer Steinwandt. *On Modeling IND-CCA Security in Cryptographic Protocols*. Cryptology ePrint Archive, Report 2003/024. https://eprint.iacr.org/2003/024. 2003 (cit. on p. 23).

[HVLT19] Mike Hamburg, Henri de Valence, Isis Lovecruft, and Arcieri Tony. *The Ristretto group*. 2019. Link. (Cit. on p. 55).

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. *Extending Oblivious Transfers Efficiently*. In: *CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. LNCS. Springer, Heidelberg, Aug. 2003, pp. 145–161 (cit. on p. 42).

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. *Zero-knowledge from secure multiparty computation*. In: *39th ACM STOC*. Ed. by David S. Johnson and Uriel Feige. ACM Press, June 2007, pp. 21–30 (cit. on p. 49).

[IR01] Gene Itkis and Leonid Reyzin. *Forward-Secure Signatures with Optimal Signing and Verifying*. In: *CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. LNCS. Springer, Heidelberg, Aug. 2001, pp. 332–354 (cit. on p. 32).

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. *Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently*. In: *ACM CCS 2013*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 955–966 (cit. on p. 46).

[JKRS20] Tibor Jager, Eike Kiltz, Doreen Riepel, and Sven Schäge. *Tightly-Secure Authenticated Key Exchange, Revisited*. Cryptology ePrint Archive, Report 2020/1279. https://eprint.iacr.org/2020/1279. 2020 (cit. on p. 68).

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. *On the Security of TLS-DHE in the Standard Model*. In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 273–293 (cit. on p. 39).

[JMM18] Daniel Jost, Ueli Maurer, and Marta Mularczyk. *Efficient Ratcheting: Almost-Optimal Guarantees for Secure Messaging*. Cryptology ePrint Archive, Report 2018/954. https://eprint.iacr.org/2018/954. 2018 (cit. on p. 71).

[JMM19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. *Efficient Ratcheting: Almost-Optimal Guarantees for Secure Messaging*. In: *EUROCRYPT 2019, Part I*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11476. LNCS. Springer, Heidelberg, May 2019, pp. 159–188 (cit. on pp. viii, 7, 71–73, 77, 79, 80).

[JS18a] Joseph Jaeger and Igors Stepanovs. *Optimal Channel Security Against Fine-Grained State Compromise: The Safety of Messaging*. Cryptology ePrint Archive, Report 2018/553. https://eprint.iacr.org/2018/553. 2018 (cit. on p. 71).

[JS18b] Joseph Jaeger and Igors Stepanovs. *Optimal Channel Security Against Fine-Grained State Compromise: The Safety of Messaging*. In: *CRYPTO 2018, Part I*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10991. LNCS. Springer, Heidelberg, Aug. 2018, pp. 33–62 (cit. on pp. viii, 7, 71–75, 77, 79, 80).

[KBB17] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. *Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach*. In: *2017 IEEE European Symposium on Security and Privacy (EuroSP)*. 2017, pp. 435–450 (cit. on p. 32).

[Kil92] Joe Kilian. *A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract)*. In: *24th ACM STOC*. ACM Press, May 1992, pp. 723–732 (cit. on p. 47).

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. *Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures*. In: *ACM CCS 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM Press, Oct. 2018, pp. 525–537 (cit. on pp. 51, 144, 145).

[KR03] Anton Kozlov and Leonid Reyzin. *Forward-Secure Signatures with Fast Key Update*. In: *SCN 02*. Ed. by Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano. Vol. 2576. LNCS. Springer, Heidelberg, Sept. 2003, pp. 241–256 (cit. on p. 32).

[Kra00] Hugo Krawczyk. *Simple Forward-Secure Signatures From Any Signature Scheme*. In: *ACM CCS 2000*. Ed. by Dimitris Gritzalis, Sushil Jajodia, and Pierangela Samarati. ACM Press, Nov. 2000, pp. 108–115 (cit. on p. 32).

[Kra05] Hugo Krawczyk. *HMQV: A High-Performance Secure Diffie-Hellman Protocol*. In: *CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. LNCS. Springer, Heidelberg, Aug. 2005, pp. 546–566 (cit. on pp. 32, 35, 37).

[Kra10] Hugo Krawczyk. *Cryptographic Extraction and Key Derivation: The HKDF Scheme*. In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 631–648 (cit. on pp. 30, 31).

[Kur02] Kaoru Kurosawa. *Multi-recipient Public-Key Encryption with Shortened Ciphertext*. In: *PKC 2002*. Ed. by David Naccache and Pascal Paillier. Vol. 2274. LNCS. Springer, Heidelberg, Feb. 2002, pp. 48–63 (cit. on p. 26).

[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. 35, 66).

[MBB+15] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. *CONIKS: Bringing Key Transparency to End Users*. In: *USENIX Security 2015*. Ed. by Jaeyeon Jung and Thorsten Holz. USENIX Association, Aug. 2015, pp. 383–398 (cit. on p. 70).

[Mic94] Silvio Micali. *CS Proofs (Extended Abstracts)*. In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 436–453 (cit. on p. 47).

[MMM02] Tal Malkin, Daniele Micciancio, and Sara K. Miner. *Efficient Generic Forward-Secure Signatures with an Unbounded Number Of Time Periods*. In: *EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. LNCS. Springer, Heidelberg, Apr. 2002, pp. 400–417 (cit. on p. 32).

[MNPV99] David M'Raïhi, David Naccache, David Pointcheval, and Serge Vaudenay. *Computational Alternatives to Random Number Generators*. In: *SAC 1998*. Ed. by Stafford E. Tavares and Henk Meijer. Vol. 1556. LNCS. Springer, Heidelberg, Aug. 1999, pp. 72–80 (cit. on p. 41).

[MOV96] Alfred J. Menezes, Paul C van Oorshot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. Chap. 12, p. 496. ISBN: 0-8493-8523-7. Link. (Cit. on p. 32).

[MP13] Moxie Marlinspike and Trevor Perrin. *Advanced Cryptographic Ratcheting*. Signal web site. Nov. 2013. Link. (Cit. on p. 33).

[MP16a] Moxie Marlinspike and Trevor Perrin. *The Double Ratchet Algorithm*. White paper. Signal's web site, 2016. Link. (Cit. on pp. 7, 61, 63).

[MP16b] Moxie Marlinspike and Trevor Perrin. *The X3DH Key Agreement Protocol*. White paper. Signal's web site, 2016. Link. (Cit. on pp. 7, 61).

[MP17] Moxie Marlinspike and Trevor Perrin. "The Sesame Algorithm: Session Management for Asynchronous Message Encryption". In: *Signal's web site* (Apr. 2017) (cit. on pp. 84, 87).

[MU10] Alfred Menezes and Berkant Ustaoglu. "On Reusing Ephemeral Keys in Diffie-Hellman Key Agreement Protocols". In: *Int. J. Appl. Cryptol.* 2.2 (2010), pp. 154–158. ISSN: 1753-0563. Link. (Cit. on p. 61).

[Nit19] Anca Nitulescu. *A tale of SNARKs: quantum resilience, knowledge extractability and data privacy*. thesis. Université PSL - ENS, 2019. Link. (Cit. on p. 48).

[NY90] Moni Naor and Moti Yung. *Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks*. In: *22nd ACM STOC*. ACM Press, May 1990, pp. 427–437 (cit. on p. 23).

[Oka93] Tatsuaki Okamoto. *Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes*. In: *CRYPTO'92*. Ed. by Ernest F. Brickell. Vol. 740. LNCS. Springer, Heidelberg, Aug. 1993, pp. 31–53 (cit. on p. 42).

[OP01] Tatsuaki Okamoto and David Pointcheval. *The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes*. In: *PKC 2001*. Ed. by Kwangjo Kim. Vol. 1992. LNCS. Springer, Heidelberg, Feb. 2001, pp. 104–118 (cit. on p. 19).

[Ped92] Torben P. Pedersen. *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*. In: *CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 129–140 (cit. on p. 52).

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. *Pinocchio: Nearly Practical Verifiable Computation*. In: *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013, pp. 238–252 (cit. on pp. 47, 48, 145).

[Poi02] David Pointcheval. *Le chiffrement asymétrique et la sécurité prouvée*. HDR thesis. ENS, 2002. Link. (Cit. on p. 15).

[Pol78] J. Pollard. "Monte Carlo methods for index computation (mod p)". In: *Mathematics of Computation* 32 (1978), pp. 918–924 (cit. on p. 19).

[PPS11] Duong Hieu Phan, David Pointcheval, and Mario Strefler. *Security Notions for Broadcast Encryption*. In: *ACNS 11*. Ed. by Javier Lopez and Gene Tsudik. Vol. 6715. LNCS. Springer, Heidelberg, June 2011, pp. 377–394 (cit. on p. 27).

[PPS12] Duong Hieu Phan, David Pointcheval, and Mario Strefler. *Decentralized Dynamic Broadcast Encryption*. In: *SCN 12*. Ed. by Ivan Visconti and Roberto De Prisco. Vol. 7485. LNCS. Springer, Heidelberg, Sept. 2012, pp. 166–183 (cit. on pp. 27, 91, 98).

[PR18a] Bertram Poettering and Paul Rösler. *Asynchronous ratcheted key exchange*. Cryptology ePrint Archive, Report 2018/296. https://eprint.iacr.org/2018/296. 2018 (cit. on pp. 29, 30, 71).

[PR18b] Bertram Poettering and Paul Rösler. *Towards Bidirectional Ratcheted Key Exchange*. In: *CRYPTO 2018, Part I*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10991. LNCS. Springer, Heidelberg, Aug. 2018, pp. 3–32 (cit. on pp. viii, 7, 71–80).

[PS96a] David Pointcheval and Jacques Stern. *Provably Secure Blind Signature Schemes*. In: *ASIACRYPT'96*. Ed. by Kwangjo Kim and Tsutomu Matsumoto. Vol. 1163. LNCS. Springer, Heidelberg, Nov. 1996, pp. 252–265 (cit. on p. 46).

[PS96b] David Pointcheval and Jacques Stern. *Security Proofs for Signature Schemes*. In: *EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. LNCS. Springer, Heidelberg, May 1996, pp. 387–398 (cit. on p. 46).

[Ram99] Blake Ramsdell. *S/MIME Version 3 Message Specification*. RFC. IETF, June 1999. Link. (Cit. on p. 59).

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. *OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption*. In: *ACM CCS 2001*. Ed. by Michael K. Reiter and Pierangela Samarati. ACM Press, Nov. 2001, pp. 196–205 (cit. on p. 30).

[RCF+20] Armando Ruggeri, Antonio Celesti, Maria Fazio, Antonino Galletta, and Massimo Villari. *BCB-X3DH: a Blockchain Based Improved Version of the Extended Triple Diffie-Hellman Protocol*. In: *2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. 2020, pp. 73–78 (cit. on p. 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. 30).

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Commun. ACM* 21.2 (1978), pp. 120–126. ISSN: 0001-0782. Link. (Cit. on p. 24).

[RST19] Blake Ramsdell, Jim Schaad, and Sean Turner. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification*. RFC. IETF, Apr. 2019. Link. (Cit. on p. 59).

[Sha84] Adi Shamir. *Identity-Based Cryptosystems and Signature Schemes*. In: *CRYPTO'84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. LNCS. Springer, Heidelberg, Aug. 1984, pp. 47–53 (cit. on p. 70).

[Sho01] Victor Shoup. *A Proposal for an ISO Standard for Public Key Encryption*. Cryptology ePrint Archive, Report 2001/112. https://eprint.iacr.org/2001/112. 2001 (cit. on pp. 24, 30).

[Sho99] Victor Shoup. *On Formal Models for Secure Key Exchange*. Cryptology ePrint Archive, Report 1999/012. https://eprint.iacr.org/1999/012. 1999 (cit. on p. 38).

[Sma05] Nigel P. Smart. *Efficient Key Encapsulation to Multiple Parties*. In: *SCN 04*. Ed. by Carlo Blundo and Stelvio Cimato. Vol. 3352. LNCS. Springer, Heidelberg, Sept. 2005, pp. 208–219 (cit. on p. 26).

[Sta06] International Organization for Standardization/International Electrotechnical Commission (ISO/IEC). *Information technology - Security techniques - Encryption algorithms - Part 2: Asymmetric ciphers - 18033-2*. 2006. Link. (Cit. on p. 30).

[Sta96] Markus Stadler. *Publicly Verifiable Secret Sharing*. In: *EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. LNCS. Springer, Heidelberg, May 1996, pp. 190–199 (cit. on p. 54).

[TD17] Alin Tomescu and Srinivas Devadas. *Catena: Efficient Non-equivocation via Bitcoin*. In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 393–409 (cit. on p. 70).

[TT01] Wen-Guey Tzeng and Zhi-Jia Tzeng. *A Public-Key Traitor Tracing Scheme with Revocation Using Dynamic Shares*. In: *PKC 2001*. Ed. by Kwangjo Kim. Vol. 1992. LNCS. Springer, Heidelberg, Feb. 2001, pp. 207–224 (cit. on p. 27).

[Tuc20] Ida Tucker. *Chiffrement fonctionnel et signatures distribuées fondés sur des fonctions de hachage à projection, l'apport des groupes de classe*. 2020LYSEN054. PhD thesis. 2020. Link. (Cit. on p. 148).

[UDB+15] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. *SoK: Secure Messaging*. In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 232–249 (cit. on pp. v, 5, 59, 70, 84).

[UG15] Nik Unger and Ian Goldberg. *Deniable Key Exchanges for Secure Messaging*. In: *ACM CCS 2015*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 1211–1223 (cit. on p. 69).

[VGIK20] Nihal Vatandas, Rosario Gennaro, Bertrand Ithurburn, and Hugo Krawczyk. *On the Cryptographic Deniability of the Signal Protocol*. In: *ACNS 20, Part II*. Ed. by Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi. Vol. 12147. LNCS. Springer, Heidelberg, Oct. 2020, pp. 188–209 (cit. on p. 69).

[WBPE21] Jan Wichelmann, Sebastian Berndt, Claudius Pott, and Thomas Eisenbarth. *Help, my Signal has bad Device! Breaking the Signal Messenger's Post-Compromise Security through a Malicious Device*. Cryptology ePrint Archive, Report 2021/626. https://eprint.iacr.org/2021/626. 2021 (cit. on p. 89).

[WT99] Alma Whitten and J. Doug Tygar. *Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0*. In: *USENIX Security 99*. Ed. by G. Winfield Treese. USENIX Association, Aug. 1999 (cit. on pp. iv, 4).

[Yao82] Andrew Chi-Chih Yao. *Protocols for Secure Computations (Extended Abstract)*. In: *23rd FOCS*. IEEE Computer Society Press, Nov. 1982, pp. 160–164 (cit. on p. 46).

[YV20] Hailun Yan and Serge Vaudenay. *Symmetric Asynchronous Ratcheted Communication with Associated Data*. In: *IWSEC 20*. Ed. by Kazumaro Aoki and Akira Kanaoka. Vol. 12231. LNCS. Springer, Heidelberg, Sept. 2020, pp. 184–204 (cit. on p. 80).

[Zim95] Philip R. Zimmermann. *The Official PGP User's Guide*. Cambridge, MA, USA: MIT Press, 1995. ISBN: 0262740176 (cit. on p. 59).

List of Figures

List of Tables

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Autour de la sécurité des messageries instantanées

Mot clés : Cryptographie, messageries instantanées sécurisées, échanges de clés à cliquet, divulgation nulle

Résumé : Les applications de messagerie instantanée sécurisée, telles WhatsApp ou Signal, sont devenues incontournables pour nos communications quotidiennes. Elles apportent une sécurité caractérisée notamment par le chiffrement de bout en bout, la confidentialité persistante ou encore la sécurité après compromission. Mais ces propriétés sont généralement limitées aux communications deux à deux. L'objectif des travaux présentés ici est d'atteindre, pour les communications à partir de plusieurs appareils et pour les communications de groupe, un niveau de sécurité optimal, comparable à celui existant pour les échanges deux à deux. Dans un premier temps, nous proposons une solution multi-appareils basée sur l'échange de clé à cliquet utilisé dans Signal et largement déployé dans d'autres applications. Dans un second temps, nous nous intéressons au protocole MLS (Messaging Layer Security) qui vise à offrir une messagerie de groupe sécurisée. La sécurité de ce protocole repose notamment sur la possibilité pour chaque utilisateur de mettre à jour la clé de groupe. Telle que spécifiée actuellement, cette fonctionnalité présente une faille. Nous proposons une solution pour sécuriser le mécanisme de mise à jour, en se basant notamment sur de nouveaux protocoles de preuve à divulgation nulle.

Title: On the security of Instant Messaging

Keywords: Cryptography, Secure Messaging, Ratcheted Key Exchange, Zero-Knowledge

Abstract: Secure Instant Messaging applications (such as WhatsApp or Signal) have become unavoidable means of communication in our every day lives. These applications offer desirable security features such as end-to-end encryption, forward and post-compromise security. However, these properties are often limited to one-to-one communications. The purpose of the work presented in this manuscript is to reach, in the multi device or in the group context, an optimal level of security, comparable to the classical one-to-one communications. On the multi-device side, we propose a Multi-Device Instant Messaging protocol, based on the Ratcheted Key exchange used in Signal and widely deployed in other applications. On the group side, we are interested in the Messaging Layer Security (MLS) protocol, which aims at providing a secure group messaging solution. The security of the protocol relies in particular on the possibility for any user to update the group secrets. In its actual design, a flaw appears in this updating process. We propose a solution to secure the update mechanism, using Zero-Knowledge (ZK) as a building block. As a main contribution, we provide two different ZK protocols to prove knowledge of the input of a pseudo random function implemented as a circuit, given an algebraic commitment of the output and the input.