Strongly Private Communications in a Homogeneous Network

Antoine Guellier

ANNÉE 2017

CentraleSupélec

UNIVERSITE BRETAGNE LOIRE

N° d'ordre : 2017-05-TH

THÈSE / CENTRALESUPÉLEC
sous le sceau de l'Université Bretagne Loire

pour le grade de

DOCTEUR DE CENTRALESUPÉLEC

Mention : Informatique

Ecole doctorale 359 « Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Electronique (MATISSE) »

présentée par

Antoine Guellier

Préparée à l'UMR 6074 - IRISA (Equipe CIDRE)
Institut de Recherche en Informatique et Systèmes Aléatoires

Strongly Private Communications in a Homogeneous Network

Thèse soutenue à Rennes le 22 mai 2017

devant le jury composé de :

Carlo AGUILAR-MELCHOR
Maître de Conférences (Université de Toulouse) / *rapporteur*

Maryline Laurent
Professeur des Universités (Télécom SudParis) / *rapporteur*

Marc-Olivier KILLIJIAN
Directeur de Recherche (LAAS-CNRS) / *président du jury*

Cristina ONETE
Post-Doctorante (IRISA/INSA) / *examinateur*

Christophe BIDAN
Professeur (CentraleSupélec) / *directeur de thèse*

Nicolas PRIGENT
Expert Technique (LSTI) / *co-directeur de thèse*

Abstract

With the development of online communications in the past decades, new privacy concerns have emerged. A lot of research effort have been focusing on concealing *relationships* in Internet communications. However, most works do not prevent particular network actors from learning the original sender *or* the intended receiver of a communication. While this level of privacy is satisfactory for the common citizen, it is insufficient in contexts where individuals can be convicted for the mere sending of documents to a third party. This is the case for so-called *whistle-blowers*, who take personal risks to alert the public of anti-democratic or illegal actions performed by large organisations.

In this thesis, we consider a stronger notion of anonymity for peer-to-peer communications on the Internet, and aim at concealing the very fact that users take part in communications. To this end, we deviate from the traditional client-server architecture endorsed by most existing anonymous networks, in favor of a *homogeneous*, fully distributed architecture in which every user also acts as a relay server, allowing it to conceal its own traffic in the traffic it relays for others. In this setting, we design an Internet overlay inspired from previous works, that also proposes new privacy-enhancing mechanisms, such as the use of relationship pseudonyms for managing identities. We formally prove with state-of-the-art cryptographic proof frameworks that this protocol achieves our privacy goals. Furthermore, a practical study of the protocol shows that it introduces *high latency* in the delivery of messages, but ensures a high anonymity level even for networks of small size.

Keywords: Privacy, Anonymity, Network, Internet, Communications, Peer-to-Peer, Homogeneous, Cryptography, Provable Security, Homomorphic Encryption

Résumé

L'avènement de l'ère digitale a changé la façon dont les individus communiquent à travers le monde, et a amené de nouvelles problématiques en terme de vie privée. La notion d'anonymat la plus répandue pour les communications sur Internet consiste à empêcher tout acteur du réseau de connaître *à la fois* l'expéditeur d'un message et son destinataire. Bien que ce niveau de protection soit adéquat pour l'utilisateur d'Internet moyen, il est insuffisant lorsqu'un individu peut être condamné pour le simple envoi de documents à une tierce partie. C'est le cas en particulier des *lanceurs d'alerte*, prenant des risques personnels pour informer le public de pratiques illégales ou antidémocratiques menées par de grandes organisations.

Dans cette thèse, nous envisageons un niveau d'anonymat plus fort, où l'objectif est de dissimuler le fait même qu'un utilisateur envoie ou reçoive des données. Pour cela, nous délaissons l'architecture client-serveur couramment utilisée dans les réseaux anonymes, en faveur d'une architecture entièrement distribuée et *homogène*, où chaque utilisateur remplit également le rôle de serveur relais, lui permettant de dissimuler son propre trafic dans celui qu'il relaie pour les autres. Dans cette optique, nous proposons un nouveau protocole pour les communications de pair à pair sur Internet. À l'aide de récents outils de preuves cryptographiques, nous prouvons que ce protocole réalise les propriétés d'anonymat désirées. De plus, nous montrons par une étude pratique que, bien que le protocole induise une grande latence dans les communications, il assure un fort anonymat, même pour des réseaux de petite taille.

Mots-clés: Vie privée, anonymat, réseaux, internet, communications, pair à pair, homogène, cryptographie, sécurité prouvable, chiffrement homomorphe

Remerciements

Je remercie tout d'abord les membres du jury, à commencer par Maryline Laurent et Carlos Aguilar-Melchor, qui ont accepté de rapporter cette thèse (sans savoir, à ce moment là, qu'elle faisait 200 pages). Merci également à Marc-Olivier Killijian et Cristina Onete d'avoir accepté de faire partie du jury en tant qu'examinateurs.

Je remercie mes encadrants Christophe Bidan, Nicolas Prigent et Cristina Onete, notamment pour la liberté qu'ils m'ont laissé dans le pilotage de mon sujet de thèse, ainsi que leur soutien et compréhension dans les moments les plus difficiles. Merci aussi d'avoir cru en moi jusqu'au bout, et (presque) réussi a me convaincre que j'avais fait du bon travail ! Je souhaite souligner ma reconnaissance envers Cristina Onete, qui a notamment a été pour moi une ressource indispensable dans ma formation en cryptographie et en sécurité prouvable, et qui n'a malheureusement pas pu figurer en tant qu'encadrante officielle.

On élargit ensuite le cercle. Je remercie toute l'équipe CIDRE, ses membres actuels et passés que j'ai pu rencontrer au cours de ma thèse. J'imagine difficilement trouver équipe plus adéquate pour moi, en terme d'ambiance et d'esprit. Parmi mes aînés, je tiens en particulier à remercier Paul, Julien et Simon pour les discussions (aussi bien celles techniques que les informelles), et le soutien qu'ils m'ont apporté. En particulier, Simon ayant un sujet de thèse très proche du mien, a joué un rôle important dans l'aboutissement de ma thèse.

Ensuite, je remercie chaleureusement le Professeur Lynn Batten, de Deakin University à Melbourne, qui m'a reçu neuf mois durant au sein du laboratoire *Securing Cyberspace* dont elle est la directrice. Cette mobilité m'aura permis de découvrir de nouvelles façon de faire de la recherche, et de remarquer la raréfaction des budgets alloués à la recherche scientifique n'est pas une spécialité française. Cette mobilité a surtout été l'occasion d'étendre mon réseau. Ainsi, je remercie Veelasha Moonsamy pour m'avoir aidé à m'intégrer sur le campus, Helaine Leggat pour m'avoir invité dans divers évènement réunissant scientifiques et industriels travaillant dans le domaine de la vie privée, Lejla Batina pour m'avoir ensuite proposé une visite à l'Université Radboud de Nimègue. Je remercie également Iynkaran Natgunanathan, Anna Krasnova, and Gergely Alpár avec qui j'ai eu l'occasion de collaborer pendant et à la suite de ces mobilités.

J'arrive maintenant aux remerciements plus personnels. D'abord, je voudrais remercier Héloïse et Arthur de m'avoir permis d'avoir un pied-à-terre dans leur *domaine* aux abords de Rennes, me permettant de terminer ma thèse sans avoir à gérer plusieurs appartements. Merci également à Bob, pour son inébranlable joie communicative à chaque retour du travail. Dans la foulée, merci à Pixel, qui m'a accompagné dans mes derniers mois de rédaction qui auraient été bien solitaires autrement.

Je garde le plus important pour la fin : un immense merci à mes parents et à ma partenaire de vie, Sophie, qui ont été un soutien inconditionnel et sans qui ce magnifique feuillet (ou ce document pdf) de 200 pages n'aurait pas abouti. En particulier, merci à Sophie, qui a su jusqu'au bout supporter les hauts, les bas, et les plus bas encore, les incertitudes, *etc.*, qui m'a toujours conforté dans mes choix, et m'a indubitablement fait évoluer et progresser.

Synopsis en Français

Ce synopsis est fourni en conformité avec la loi relative à l'emploi de la langue française de 1994[1]. Il reprend la structure de la thèse, et résume les chapitres un à un.

Introduction

L'avènement de l'ère digitale et d'Internet a profondément changé la façon dont les individus communiquent à travers le monde, et amené de nouvelles problématiques de vie privée. Les sociétés se sont adaptées, et le parlement européen, dans ses directives de 1995 et 2002, a notamment reconnu la nécessité de la confidentialité et de l'anonymat des communications. Les organisation et acteurs agissant en faveur du respect de la vie privée mettent en avant la nécessité de cet anonymat pour la liberté d'expression, et, plus généralement, pour le bon fonctionnement d'une démocratie. Cependant, si l'anonymat est important pour le citoyen, c'est une nécessité pour certains individus. Dans cette thèse, nous considérons un scénario où un informateur prend des risques personnels pour communiquer à un journaliste des information révélant des actions illégales ou discriminatoires menées par des instances gouvernementales ou de grandes organisations. Dans ce cas, la protection de l'anonymat de l'informateur est cruciale. Le but de cette thèse est de proposer un protocole permettant de protéger l'anonymat des communications par Internet.

1 Contexte et Modèles

Contrairement à l'architecture *client-serveur* couramment utilisée dans le domaine des réseaux anonymes sur Internet, nous nous proposons de construire un protocole sur un réseau *homogène*. Alors que, dans l'architecture client-serveur, les individus utilisateurs du réseau dépendent de serveurs relais fournissant l'anonymat *en tant que service*, dans l'architecture homogène, les nœuds ne sont pas hiérarchisés. En effet, tous les nœuds du réseau participent en tant qu'utilisateur (envoyant et reçevant des messages), *et* en tant que relais. D'autre part, nous supposons que les connexions entre nœuds forment un *graphe de topologie* connexe mais incomplet. C'est à dire que chaque nœud est connecté à un petit nombre de *voisins* avec qui il peut échanger des messages directement. Pour permettre à un nœud d'envoyer un message à un nœud non voisin, le protocole doit donc organiser le relai du message dans ce graphe incomplet.

En terme de vie privée, le protocole vise à préserver la vie privée de ses utilisateurs, même en présence de collusions de nœuds corrompus et d'un observateur global du réseau

[1] Translation: this synopsis is provided in a accordance to the French law on written academic productions of 1994.

(capable de voir tous les messages transitant entre les nœuds du réseau). Cependant *l'adversaire* est considéré passif. C'est à dire que les nœuds corrompus ne dévient pas du protocole, mais essaient uniquement d'en apprendre le plus possible sur le réseau et les autres nœuds en participant au relai des messages.

Le protocole vise à réaliser l'anonymat de l'envoyeur, l'anonymat du receveur, et à résister aux attaques basées sur l'analyse de trafic. En vue de l'adversaire considéré, nous définissons l'anonymat de l'envoyeur comme l'impossibilité même de détecter le fait qu'un nœud envoie un message dans le cadre d'une communication. De même, l'anonymat du receveur est défini comme l'impossibilité de détecter le fait qu'un nœud reçoive un message. Ces propriétés sont donc plus fortes que la plupart des travaux existants, qui ne visent qu'à cacher qui communique avec qui, mais considèrent acceptable de laisser certains acteurs du réseau apprendre l'identité de l'envoyeur ou du receveur d'un message (tant que les deux ne sont pas connues simultanément).

2 Outils cryptographiques

Ce chapitre est l'occasion de présenter les outils cryptographiques utilisés dans la thèse. Premièrement, nous faisons usage du chiffrement, dans ses deux principales variantes : le chiffrement à clé publique, et le chiffrement à clé secrète. Un chiffrement d'un message m avec la clé publique pk est noté, $\mathsf{Chiff}(pk, m)$, alors que le chiffrement avec une clé secrète k est noté $\{m\}_k$. Nous faisons aussi usage de fonctions de hachage (le récent standard de NIST nommé SHA-3, en l'occurrence), et du protocole d'échange de clé de Diffie-Hellman.

Cependant, la particularité du protocole réside dans l'utilisation du *chiffrement homomorphe* et du *re-chiffrement*. Plus exactement, nous utilisons le schéma de chiffrement homomorphe de Elgamal, qui permet notamment, à partir de deux chiffrés $c_1 = \mathsf{Chiff}(pk, m_1)$ et $c_2 = \mathsf{Chiff}(pk, m_2)$, de calculer le chiffré $c = \mathsf{Chiff}(pk, m_1 \cdot m_2)$ du produit $m_1 \cdot m_2$. Cette propriété permet en effet d'effectuer des calculs sur des données chiffrées. D'autre part, le schéma de Elgamal supporte l'opération de *re-chiffrement*, qui permet de *modifier l'apparence d'un chiffré*. Plus exactement cette opération prend en entrée un chiffré $c = \mathsf{Chiff}(pk, m)$ et produit un chiffré $c' = \mathsf{Chiff}(pk, m)$ méconnaissable de c, tout en assurant la confidentialité du message m durant le processus. Cela permet en particulier de modifier l'apparence d'un chiffré au cours de son relai à travers le réseau, de manière à ce qu'il soit impossible de suivre sa progression (du moins, pas trivialement).

3 État de l'art

Avant de présenter notre protocole, nous passons en revue les travaux existants dans le domaine des communications anonymes sur Internet. Dans la littérature, les protocoles anonymes sont souvent répartis selon la latence qu'ils introduisent dans les communications (comparé à un simple paquet IP directement communiqué par l'envoyeur au receveur). Nous rajoutons une troisième catégorie, identifiée aux protocoles *homogènes*, *i.e.* ceux supposant une architecture homogène.

Les protocoles à *faible latence* sont les plus efficaces et les plus populaires. Le protocole

Tor, utilisé aujourd'hui par plus de deux millions d'individus, appartient à cette catégorie. Cependant, ce sont aussi les protocoles les moins robustes aux attaques contre l'anonymat. En effet, l'approche des protocoles à faibles latence consiste à intégrer tout mécanisme protégeant l'anonymat des communicants *tant que ceux-ci ne dégradent que peu les performances du réseau.* En conséquence, un protocole à faible latence ne se protège pas (ou très peu) contre les attaques basées sur l'analyse de trafic. Au minimum, un protocole à faible latence se contente de modifier l'apparence des messages entre chaque nœud relais pour empêcher leur traçage (en utilisant le re-chiffrement, ou, plus souvent, le chiffrement en structure *d'oignon*). Mais, dans Tor en particulier, si le premier et le dernier serveur relais sont corrompus, il est possible de complètement casser l'anonymat (*i.e.* il est possible de savoir qui communique avec qui).

En comparaison, les protocoles à *forte latence* intègrent nativement une protection contre l'analyse de trafic, mais sont moins efficaces. En effet, ces protocoles sont fait pour des applications *non interactives*, comme l'échange d'emails, mais ne supportent pas la consultation de sites web ou le transport d'une session ssh par exemple. Une des principales approches pour résister à l'analyse de trafic est de fonctionner en intervalles de temps discrets (appelées *tours*) : chaque nœud relais accumule les messages qu'il reçoit pendant un tour, et les envoie tous d'un seul tenant et dans un ordre aléatoire à la fin du tour vers le prochain nœud relais. Ce mécanisme est nommé *mixage* des messages. Combiné avec la modification de l'apparence des messages à chaque nœud relais, ce mécanisme rend le traçage des messages beaucoup plus difficile. Cependant, si implémentés dans une architecture client-serveur, ces protocoles révèlent tout de même l'identité des envoyeurs et des receveurs. C'est à dire que, comme dans les protocoles à faible latence, seules les *relations de communications* sont dissimulées. D'autre part, il existe une attaque spécifique aux protocoles employant le *mixage* de messages, permettant dans certains contextes de retrouver exactement quel envoyeur communique avec quel receveur. Il suffit pour cela d'observer, sur plusieurs tours, quels envoyeurs et receveurs participent à chaque tour de mixage.

En vue de ces résultats et des objectifs en terme d'anonymat que nous avons posés, nous nous tournons donc vers l'architecture *homogène*, dans laquelle l'observation des envoyeurs et receveurs peut être empêchée, et où l'attaque mentionnée précédemment ne peut être menée. La littérature sur les réseaux anonymes ne comporte que peu d'exemples de protocoles homogènes. Le plus emblématique est le protocole Tarzan, reposant sur un modèle de réseau similaire au nôtre : une architecture homogène et un graphe de topologie incomplet. Les auteurs du protocole remarquent que, contre un observateur global du réseau, une architecture homogène ne suffit pas en elle même à empêcher la détection des envoyeurs et receveurs. Ils proposent, en complément, un mécanisme basé sur l'utilisation de faux messages et la limitation du trafic des nœuds. Ainsi, Tarzan fait un pas vers la réalisation de l'anonymat des envoyeurs tel que nous le définissons. Cependant, c'est un protocole à faible latence, donc susceptible à l'analyse de trafic, et qui ne protège pas les receveurs.

4 Le protocole anonyme

Dans ce chapitre présentant la principale contribution de cette thèse, nous construisons un protocole dans la continuation de l'état de l'art. Inspiré de Tarzan, il intègre également des mécanismes adaptés des protocoles à forte latence afin d'empêcher l'analyse de trafic. Plus précisément, nous menons une analyse poussée permettant d'implémenter les mécanismes de faux messages et de limitation de trafic de Tarzan de manière plus robuste. Puis, nous les intégrons avec l'idée de *mixage* des messages. Pour modifier l'apparence des messages, nous mettons en avant une utilisation du re-chiffrement avec le schéma de Elgamal. Le résultat est un protocole à forte latence et homogène, dans lequel l'anonymat ne provient pas d'une entité centrale ou de *serveurs* de relais, mais de la volonté des nœuds à s'entraider. En effet, par construction, plus un nœud fournit de trafic à ses voisins pour camoufler leurs propres communications, plus ceux-ci peuvent l'aider en retour.

Enfin, notre protocole s'adresse à des usagers nécessitant de fortes garanties d'anonymat, et prêts à payer le prix de cet anonymat. Aussi, il ne permet pas d'accéder à des sites web, mais supporte uniquement des communications entre pairs prenant activement part au réseau. Cette application contraste avec les protocoles anonymes déployés et utilisés activement aujourd'hui, qui visent à fournir un anonymat minimal pour tout usager d'Internet, et qui sont principalement utilisés pour consulter des sites web externes au réseau anonyme.

En plus de ces éléments, le protocole propose également un nouveau moyen de gérer les identités des nœuds dans le réseau, en utilisant des *pseudonymes de relation*. C'est à dire que chaque nœud a autant d'identités qu'il y a d'autres nœuds dans le réseau : un nœud donné est désigné sous un pseudonyme différent par chaque autre nœud dans le réseau. Le pseudonyme utilisé par le nœud X pour désigner le nœud Y est noté $PS_{X \to Y}$. L'utilisation de ce type de pseudonymes a plusieurs avantages : ceux-ci permettent à un receveur de rester anonyme même vis-à-vis de l'envoyeur (réalisant ainsi un équivalent des *services cachés* de Tor), et réduisent l'impact d'une potentielle dé-anonymisation. En effet, si un certain nœud corrompu X parvient à trouver l'identité de l'utilisateur du réseau qui se cache derrière un pseudonyme $PS_{X \to Y}$, il ne peut pas diffuser cette information à d'autres parties. Plus exactement, l'information « $PS_{X \to Y}$ *désigne en fait* Y » n'est d'aucune utilité pour les autres nœuds du réseau : les pseudonymes sont construits pour être cryptographiquement sûrs, de sorte que les pseudonymes $PS_{X \to Y}$ et $PS_{X' \to Y}$ utilisés par deux nœuds X et X' distincts pour désigner Y ne sont pas *chaînables* entre eux.

L'utilisation de ces pseudonymes a un impact sur la construction du protocole. Premièrement, cela nécessite de recourir à une phase de *découverte du réseau*. C'est à dire que, avant de communiquer, les nœuds doivent échanger des informations, afin d'apprendre la topologie du réseau et les pseudonymes des nœuds qui le constituent. Cette approche contraste avec la plupart des protocoles existants (dont Tor) : alors que ceux-ci construisent des routes éphémères en partant de l'envoyeur, nous construisons des routes durables en partant des receveurs. Cette approche permet d'obtenir des routes partagées par *plusieurs envoyeurs* (chose impossible avec des routes éphémères). En outre, elle a l'avantage de ne pas nécessiter de serveur central qui, traditionnellement,

donne aux nœuds les informations à propos du réseau. Cette phase de découverte du réseau est construite à partir de *propositions de route*, unité d'échange qui permet à un nœud d'annoncer à ses nœuds voisins qu'il est capable de relayer les messages de ces derniers vers un receveur. Ledit receveur est désigné par des pseudonymes pour rester anonyme, et, plus généralement, les propositions de route sont construites de sorte à donner le minimum d'information sur les routes (puisque ces informations permettraient par la suite de monter des attaques contre l'anonymat). Cela est réalisé par l'utilisation du chiffrement homomorphe, permettant de manipuler les informations sur les routes à l'intérieur de chiffrés.

Une seconde conséquence découlant de l'utilisation de pseudonymes est la nécessité d'introduire un mécanisme *d'initialisation de communication*. C'est à dire que, pour permettre à un informateur de trouver un journaliste spécifique dans le réseau, il faut lui permettre de traduire l'identité d'un individu en un pseudonyme valide dans le réseau, afin ensuite de trouver une route vers ledit individu. La difficulté est cependant de réaliser cette fonctionnalité sans briser ni les propriétés des pseudonymes, ni les propriétés d'anonymat. La solution proposée consiste à utiliser un nœud intermédiaire, qui aidera l'informateur à trouver le journaliste souhaité dans le réseau. Dans cette opération, le nœud intermédiaire n'apprend pas l'identité du journaliste, tandis que l'informateur n'apprend pas le pseudonyme du journaliste (et ainsi, ne brise pas l'anonymat fournit par les pseudonymes).

5 Preuves de sécurité et de vie privée

Après avoir présenté notre protocole, nous l'étudions sous l'angle de la *sécurité prouvable*, et prouvons formellement ses propriétés de vie privée et de sécurité. Dans un premier temps, nous étudions les propriétés cryptographique des pseudonymes, montrant ainsi qu'ils remplissent leur rôles et dissimulent l'identité des nœuds qu'ils désignent. Ensuite, nous étudions le protocole dans son entièreté. Pour cela, nous utilisons deux *frameworks* complémentaires : le framework de composition universelle (UC) et le framework AnoA. Le premier permet de faire apparaître les propriétés de base du protocole et le transforme en un objet plus facilement manipulable dans les preuves cryptographiques. Dans une seconde phase, il est ainsi plus aisé de prouver *e.g.* l'anonymat des envoyeurs en utilisant AnoA. Cette approche de preuve en deux étapes est courante dans les preuves de protocoles anonymes. Ces derniers étant des objets complexes (en comparaison de *petits* protocoles cryptographique), ce découpage permet de simplifier les preuves formelles.

Cependant, dans notre protocole, tout n'est pas prouvable par les outils que fournit la cryptographie aujourd'hui. En premier lieu, aucune méthode connue ne permet de prouver formellement la résistance à l'analyse de trafic. Ainsi, les mécanismes empêchant l'analyse de trafic, tels que l'utilisation de faux messages et le *mixage* de messages, ne peuvent être inclus dans les preuves. Pour contourner cette difficulté, qui empêcherait toute tentative de preuve, nous supposons qu'un observateur du réseau ne peut pas effectuer d'analyse de trafic, mais que, si des nœuds corrompus sont présents dans le réseau, ceux-ci en sont capables. A partir de cette hypothèse, l'approche pour les preuves d'anonymat de l'envoyeur et du receveur est d'abord de quantifier la probabilité que,

sur une route, il y ait au moins un nœud corrompu. Dans l'éventualité où aucun nœud corrompu ne se trouve sur la route, l'anonymat peut être prouvé parfait. Dans le cas échéant, une analyse est nécessaire pour quantifier la probabilité que ces nœuds corrompus retrouvent l'envoyeur et/ou le receveur. Cependant, la complexité du protocole rend difficile une analyse en profondeur avec les outils de preuve disponibles aujourd'hui. Nous choisissons donc l'approche conservatrice, standard dans le domaine de la sécurité prouvable, et supposons que la présence d'un ou plusieurs nœuds corrompus sur la route implique un anonymat nul (*i.e.* que les nœuds corrompus trouvent systématiquement l'envoyeur et le receveur avec probabilité 1).

Le résultat de ce chapitre est donc une sous approximation de l'anonymat réellement fourni par le protocole. En effet, les preuves ne montrent pas que si un nœud corrompu se trouve sur une route, l'anonymat des communicants utilisant cette route est immédiatement cassé. Au contraire, plusieurs éléments semblent indiquer que ce n'est pas le cas, du moins en général (voir notamment le prochain chapitre). Cependant, prouver ce fait semble demander des hypothèses fortes sur le réseau, et nécessite de modéliser la forme du trafic (tâche pour laquelle aucune fondation théorique n'existe actuellement). Les preuves proposées représentent cependant un premier pas vers une analyse formelle complète du protocole.

6 Implémentation : performances et vie privée en pratique

Le chapitre précédent étudie le protocole sur le plan théorique. Celui-ci l'étudie sur le plan pratique. Nous présentons une implémentation préliminaire du protocole, en utilisant un *simulateur à évènement discret*. A savoir, l'implémentation est réalisée en Python avec la librairie SimPy. L'idée est d'obtenir un code permettant de mener des *simulations* du protocole, afin de mesurer ses performances, l'impact de ses divers paramètres, et le niveau d'anonymat fourni en pratique.

Les résultats en terme de performances montrent des délais dans les communications semblables à d'autres protocoles à forte latence proposés par le passé. La communication d'un message d'un envoyeur (informateur) à un receveur (journaliste) prend en moyenne 15 minutes[2]. La découverte du réseau, elle, prend jusqu'à 24 heures. Cependant, cette étape préliminaire n'est effectuée qu'une unique fois en début de vie du réseau. Des mesures complémentaires montrent que ces latences dans les communications et la découverte du réseau s'expliquent principalement par les mécanismes mis en place pour se prémunir contre l'analyse de trafic : le *mixage* et le fonctionnement en *tours*, ainsi que les faux messages et la limitation du trafic des nœuds.

Pour mesurer l'anonymat, nous proposons une méthodologie adaptée de métriques préexistantes, notamment en comblant leurs lacunes connues. L'idée est de mesurer, en pratique, la probabilité qu'une collusion de nœuds corrompus sur une route devine correctement l'identité de l'envoyeur et/ou du receveur. Les résultats montrent que, même avec 60% de nœuds corrompus dans le réseau, la probabilité pour les nœuds corrompus de deviner correctement l'identité de l'envoyeur *ou* celle du receveur est

[2]Coût amorti pour un message au sein d'une session de 40 messages.

inférieure à 0.2. Ces résultats sont valables pour un réseau de *petite taille* (100 nœuds), et l'anonymat augmente avec le nombre de nœuds présents dans le réseau (à ratio de corruption constant). En comparaison, une récente étude du protocole Tor montre que cette probabilité de dé-anonymisation est atteinte pour seulement 0.33% de nœuds corrompus. Ces résultats empiriques sont beaucoup plus optimistes que les résultats théoriques issus du précédent chapitre, qui ne donnent qu'une sous approximation de l'anonymat effectif, et laissent espérer qu'une future analyse formelle approfondie donnera des résultats satisfaisant.

Conclusion

Au cours de cette thèse, nous avons proposé un nouveau protocole préservant l'anonymat des communications sur Internet, et avons validé ce travail à travers une approche formelle ainsi qu'une étude pratique de ses propriétés. Ce protocole fournit un anonymat plus fort que la plupart des travaux passés, et montre des performances acceptables. Dans de futurs travaux, plusieurs axes d'amélioration sont envisageables. En particulier, il est nécessaire de considérer la sécurité contre des nœuds non plus passifs mais *actifs* (*i.e.* des nœuds pleinement malveillants), et il est possible d'augmenter encore le niveau de vie privée, en tentant de cacher le fait même qu'un nœud prenne part au réseau anonyme.

Ce travail s'inscrit dans les débats actuels, ayant cours notamment depuis les révélations d'Edward Snowden en 2013. En vue du risque encouru par les lanceurs d'alerte apparus dans les médias ces dernières années, notre protocole est une solution permettant de garantir que la communication de documents et d'informations sensibles par ceux-ci ne seront même pas détectés. De plus, notre protocole est tout à fait adapté à ce genre de scénario, et à une organisation basée sur une communauté d'activistes défendant la vie privée. En effet, il suppose des utilisateurs prêts à dédier des ressources non négligeables pour assister des *informateurs*, et fournit de l'anonymat même pour de petits réseaux. Aussi, notre protocole est principalement fait pour des communications entre pairs prenant activement part au réseau anonyme, contrairement aux protocoles utilisés activement aujourd'hui, qui visent plutôt à fournir un anonymat minimal pour tout usager d'Internet.

Contents

List of Figures

List of Tables

Introduction

General Context

The advent of the digital age and of the Internet in the late twentieth-century has brought new technologies that have deeply moved the way individuals communicate across the world. These technical improvements have had great societal, political, and economical consequences. In particular, in the last decades, privacy in online communications has been rising as a major concern. Individuals now store and communicate over the Internet massive amounts of personal information every day, that is processed by both public and private actors. Societies have to adapt to these evolutions, and in particular ensure the protection of personal data from theft, misuse, or disclosure. In this regards, the European Union legal framework (Directives 1995/45/EC [Eur95], 2002/58/EC [Eur02], soon to be replaced by 2016/679 [Eur16]) recognises that the protection of personal data is a fundamental right, and declares that:

> "Member States shall ensure the confidentiality of communications and the related traffic data [...]. In particular, they shall prohibit listening, tapping, storage or other kinds of interception or surveillance of communications and the related traffic data by persons other than users, without the consent of the users concerned." (Directive 2002/58/EC, Article 5(1))

Not only the *contents* of communications (the data itself), but also the so-called *meta-data* of these communications, are considered as personal data. Indeed, the identity of communicants (*i.e.* individuals who take part in a communication) is an information that can be as sensitive as the data they exchange. For instance, the fact that a web user connects to the AA.com website is a sensitive information, that this user may not want to see disclosed.

However, in recent years, many countries have been promulgating laws that go against these principles. To take the case of France, new laws were voted in 2015 to extend the surveillance capabilities of the police and intelligence agencies [Fre15]. More recently, the United Kingdom passed the *Investigatory Powers Bill*, granting intelligence agencies unprecedented capacities for mass surveillance, in particular on mobile communications [Uni16]. The situation in the United States is similar, as revealed in 2013 by Edward Snowden. This former NSA contractor disclosed, with the help of the journalists Glenn Greenwald and Laura Poitras, the mass surveillance programs carried out by the NSA in the United States of America and around the world [Gre14]. In the wake of this latter event (in particular), public debates have divided privacy advocates and governments officials. While the former reject the mass surveillance of individuals (preferring legitimate targeted surveillance), state bodies put forward the need for a trade-off

between state security and individual privacy, especially in light of the context of a looming terrorist threat. Yet, we advocate that privacy in communications is a necessity, even for the citizen with "*nothing to hide*" [Sol07]. Indeed, mass surveillance has the pervasive effect of modifying individuals' behavior, or at the very least to hover in their mind as they take actions in the virtual world. Ultimately, it can be argued that mass surveillance endangers freedom of speech, critical thinking, and the formation of different opinions, which are all fundamental concepts for democracy [Sol07; Gre14; Rog15].

Motivating Use-Case

If privacy is indeed necessary for common citizens, it is utterly critical for individuals or organisations residing in authoritarian regimes, or non-authoritarian ones that *e.g.* carry out operations putting democracy at risk. In this thesis, we consider a scenario in which an individual, hereby called an *informant*, deliberately breaks the law of the country she inhabits, in order to divulge illegal or immoral practices (or, at least, immoral in her opinion) carried out by her government or state officials. For that, we consider that this informant is willing to communicate a set of resources to a *journalist* (or a human rights organisation, or any party that can safely reach a greater public). This scenario is of course inspired from the case of Edward Snowden, but also reflects the story of other *whistle-blowers* such as Antoine Deltour in the so-called *LuxLeaks* case [Glo16], or Chelsea Manning, who revealed torture practices of the US army in Iraq [Sle13]. Although, in all these examples, the general public deemed the actions of the whistle-blowers as legitimate and contributing to the public good, they were prosecuted, and convicted more often than not. In this thesis, we thus aim at protecting individuals in constraining contexts.

Previous Works

In this context, we believe that technology can empower individuals and provide the necessary privacy protections. Anonymity and privacy in Internet communications is the subject of a large body of literature [Fre17]. Depending on the exact definition of anonymity that is considered, the privacy guarantees differ from protocol to protocol. Furthermore, there are several ways to achieve the same functionality. However, a common point to most works in anonymous networking is the base idea originally formulated by Chaum in 1981 [Cha81]: to introduce *indirections* between the two communicating users. That is, instead of Alice directly sending its messages to Bob, an anonymous network relays Alice's messages over several hops before delivering them to Bob.

This base idea, implemented in its most simple form, only conceals the identity of the initiator of the communication (Alice, in the example) to the receiver at the other end (Bob). However, many other stronger notions of anonymity were proposed over the years. In particular, the now well-known Tor network [DMS04], currently serving over two million users, conceals communication *relationships* (*i.e.* who communicates with whom), even to network observers and relay nodes in the anonymous network. Tor belongs to the category of *low latency* protocols, which aim at minimizing the overhead introduced by the *indirections*, so as to yield a network supporting *interactive* and

latency-sensitive applications such as web browsing. However, low latency protocols are, by construction, susceptible to de-anonymisation of communicants through traffic analysis. In contrast, *high latency* protocols, such as mixnets [DP04], are less efficient, but more robust to de-anonymisation. Basically, to thwart traffic analysis attacks, mixnets tamper with the flow of messages, by delaying them and/or changing the order in which they are delivered. However, both the Tor and mixnet approaches fail to ensure a level of anonymity sufficient for the use-case considered in this thesis. Indeed, they are both based on a client-server architecture, where the network users are merely clients using the anonymous network (composed of relay servers) *as a service*. As a consequence, the first relay server automatically learns the identity of the initiator of any given communication, and the last one learns the identity of the corresponding receiver. Said otherwise, these types of networks only ensure that no single entity can know the initiator and receiver *at the same time*, but do not prevent them from learning one of the two.

Although the level of anonymity provided by a client-server architecture may be sufficient in many cases, the fact that the communicants' identities can be uncovered is an issue in our informant-journalist scenario. Actually, in this scenario, the very fact that the informant is communicating should be concealed. To achieve this stronger version of anonymity, a few works propose to depart from the client-server architecture, preferring to endorse what we hereby denote as a *homogeneous architecture*, in which each node is a client and a server at the same time [FM02]. That is, every node is a user of the network, but also relays messages for other nodes. With additional mechanisms (or by introducing some assumptions), in a homogeneous architecture, it is possible to prevent the very detection of message sending (and receiving as well). Indeed, since every node relays messages for other nodes, even the first relay after the initiator of a communication can not deduce with certainty whether the latter is the actual sender of the messages, or a simple relay.

However, in practice, existing protocols endorsing a homogeneous architecture present other weaknesses, and do not provide the level of anonymity we aim for. In particular, most of them are low latency ones, and thus fail to provide anonymity against traffic analysis attacks.

Approach and Contributions

The work presented in this thesis is in continuation of previous works on homogeneous networks. Our goal is to design a fully distributed Internet overlay, that ultimately ensures anonymity of communicants and prevents the detection of the very fact that a node communicates. Furthermore, these properties should hold even in the presence of a global network observer, in the presence of (collusions of) corrupted nodes, and resist to traffic analysis attacks. This level of anonymity is stronger than in past works aiming at ensuring anonymity over the Internet.

To achieve these goals, we start from Tarzan, a homogeneous protocol proposed by Freedman and Morris [FM02]. As the authors note, to prevent the detection of communicating nodes against a global network observer, such an architecture is not sufficient. Tarzan thus additionally proposes mechanisms based on the limitation of

traffic rates, and the sending of *dummy* messages (*i.e.* fake messages that do not actually contain any data). We propose a stronger version of these mechanisms to achieve our desired level of anonymity. The result is an anonymous network in which privacy does not stem from central entities (or relay servers), but from the willingness of nodes to help each other in staying anonymous. By design, the more a node helps its neighbors with cover traffic, the more those can help it in return.

The protocol we propose introduces several other new mechanisms and defenses. First, we adapt some techniques proper to mixnets into a homogeneous architecture, so as to prevent the possibility of traffic analysis. Secondly, we propose and study the use of *relationship pseudonyms* [PK01] in anonymous networking. That is, any given node in the network is known by each other node under a different pseudonym. These pseudonyms are designed to be cryptographically secure, implying that they conceal the identity of the node they designate. One advantage of these *relationship* pseudonyms is that they ensure a clear separation of knowledge between nodes. This can be seen as a measure of *damage control*: if a malicious network actor de-anonymises a given node, the pseudonym's properties prevent her from sharing her knowledge with other malicious actors. Using relationship pseudonyms for anonymous networking represents a drastic change in identity management compared to previous work, and raises new challenges. One consequence is that, contrarily to the more traditional construction of network routes *on-the-fly* (when a new communication is initiated), the proposed protocol requires a phase of network discovery, and builds long-lived routes. A third notable characteristic of the protocol is its heavy use of homomorphic encryption, a cryptographic primitive that allows to make computations on encrypted data. With this tool, we implement the computation of information about the routes (and the computation of the pseudonyms) in a way that limits the leaking of information about the nodes composing these routes.

Finally, we conduct a thorough formal study of the designed protocol, under the angle of *provable security*. That is, based on the cryptographic properties of the various primitives that we use, we prove that our protocol achieves the desired privacy properties. Note that producing these cryptographic proofs is a contribution in itself, since the formal study of fully-fledged anonymous protocols remains quite challenging with the currently available tools. Finally, we implement a *proof-of-concept* version of the protocol, and study its performances and practical anonymity. Results show that the protocol introduces high latency, comparably to mixnets, but ensures strong anonymity even for small networks (*i.e.* with a few hundred nodes). All these elements show that the proposed protocol fits into our informant-journalist scenario.

Organisation of the Thesis

This thesis is made of six chapters. In Chapter 1, we present the context of the thesis. In particular, we define terms pertaining to anonymous communications networks. Then, we present the system and adversary models considered in this thesis, before detailing and discussing the privacy properties we aim to ensure. Chapter 2 is an informal introduction to existing cryptographic primitives that are used as *building blocks* in anonymous networking, and in our protocol in particular. To review the existing works in private

communications, Chapter 3 expands on the elements presented in this introduction, and distinguishes protocols according to the latency (high or low) they introduce, and to the architecture (client-server or homogeneous) they assume. This chapter also proposes a review of known attacks against privacy in anonymous networks, along with existing counter-measures. Chapter 4 represent the core of this thesis, where we detail our new protocol for strongly private communications over the Internet, and explain the role of each component of the protocol. The two subsequent chapters study and analyse this protocol. In Chapter 5, we prove that the protocol achieves the properties laid out in Chapter 1. Finally, Chapter 6 presents a proof-of-concept implementation of the protocol, along with the results of network simulations aimed at studying its efficiency and practical privacy. In the conclusion, we present a summary of our contributions in the field of anonymous communications, and propose leads for further improvements. We also summarise the new perspectives for anonymous communications that our work puts in light, and the lessons learned.

1. Context

This chapter presents the general context of the thesis. First, the relevant technical terms and concepts are defined. With this terminology, the system and adversary models are laid out. Finally, the privacy properties to be ensured in the anonymous communication protocol are presented along with other side goals.

1.1. Terminology

A communication protocol involves an heterogeneous collection of *nodes* that are willing to communicate to each other. Some nodes may be *clients*, others may be *servers*, some may be both at the same time. An *anonymous* communication protocol enables communication between client nodes while ensuring some form of anonymity or privacy to its users. The nodes taking part in anonymous communication protocol collectively form an *anonymous network*.

Definition 1 (Anonymous Network). *An anonymous network is a collection of nodes running* specific software *in order to participate in an anonymous communication protocol.*

Definition 2 (User). *A user of an anonymous network is an entity (*e.g. *an individual, group of individuals, or organisation) seeking to obtain anonymity or privacy from the network.*

Definition 3 (Client). *A client in an anonymous network is a node run by a* user.

Definition 4 (Server). *A server in an anonymous network is a node enabling or aiding* clients *in obtaining anonymity or privacy.*

Any anonymous network assumes a topology graph, where nodes are vertices and edges represent a direct communication link between two nodes. In this thesis, direct communication links are assumed bidirectional (*i.e.* the graph is *undirected*). This topology graph

may be *complete* or *partial*, but is always *connected*. The direct communication links between nodes are usually realised through an underlying, non-anonymous, pre-existing network enabled by standard protocols such as TCP/IP or Ethernet. This latter network is denoted the *underlay*.

Definition 5 (Underlay, Overlay). *The underlay is the network on which the anonymous network is based, functioning with a standard communication protocol of its own. Conversely, the anonymous network is sometimes denoted as a network overlay (e.g. an Internet overlay).*

The nodes directly accessible from a given node in the anonymous network's topology graph are its *neighbors*.

Definition 6 (Neighbors). *The neighbors of a given node are the nodes with which it has a direct communication link. That is, the nodes with which it shares an edge in the* topology graph.

Generally speaking, the goal of an anonymous network is to allow users to communicate anonymously, *i.e.* to conceal which sender sends messages to which receiver. To do so, a typical technique, first presented in the seminal work of Chaum [Cha81], is to introduce *indirections* on the path taken by a message. Thus, a message may be relayed over several hops in the anonymous network, and can be seen either as a sequence of *link messages*, or as one *end-to-end message*.

Definition 7 (Link & End-to-End Message). *Messages generally designate any data or bytes exchanged between nodes. A link message is a message from a node to one of its neighbors in the topology graph. An end-to-end message is a message relayed over several hops in the anonymous network, from its sender to its receiver. A link message is said to carry a particular end-to-end message.*

To avoid confusion between the action of sending an end-to-end message as the original sender of a communication, and the action of sending a link message so as to relay the end-to-end message it carries, the terms of *end-sender* and *link-sender* are introduced (and similarly for receivers).

Definition 8 (End-sender, End-receiver, Relay). *The original sender of an* end-to-end message *is called an end-sender, performing the action of* end-sending*, while the term link-sender designates the sender of a* link message*. Analogously, the distinction is made between end-receiver and link-receiver. For a given communication, a relay is a node participating in transporting messages from an end-sender to an end-receiver: it* link-receives *and* link-sends *messages w.r.t. this communication.*

The goal of an anonymous network is usually to protect *end-senders* and *end-receivers*, the end users of a *communication session*.

Definition 9 (Communication Session). *A communication session between an* end-sender S *and an* end-receiver R *consists in the end-sending of a set of* end-to-end messages *by S to R.*

Note that a given node may be end-sender with regards to some communication session, and end-receiver or relay with regards to another. When clear from the context, the term *sender* is used indifferently to designate a link-sender or end-sender.

Depending on the semantics of its content, an end-to-end message is a *payload* or a *routing* message. The former contains application-layer data, while the latter contains routing information communicated among nodes in order to make the protocol work.

Definition 10 (Payload & Routing Message). *A payload message is an* end-to-end message *containing application data. A routing message is an* end-to-end message *containing information necessary to make the anonymous communication protocol function.*

Note that the goal of an anonymous network is to ultimately protect the exchange of *payload messages* in communication sessions, the *routing messages* being a means to this end. However, if observing routing messages can lead later to a breach in privacy, routing messages must also be protected.

A given node has different identities depending on the observed network layer. For instance, if the underlay network is the Internet, the node's *underlay identity* is its IP address. The user running the node determines its *real-world identity*. It may be an individual, an organisation, or a company. Finally, a node may additionally have an identity in the anonymous network, its *anonymous network identity*. The term address is sometimes used as an alias for *identity*. Ultimately, what needs to be protected by the anonymous communication protocol is the real-wold identity of the node, or more accurately, the link between it and the node's actions.

Definition 11 (Real-world, Underlay, and Anonymous network Identities). *The real-world identity of a client node is the identity of the user running the node. A node also has an underlay identity, relevant to the protocol run in the underlay (e.g. its IP or MAC address). Additionally, a node may have an anonymous network identity, an identity only meaningful in the anonymous communication protocol.*

1.2. System and Communication Model

In this work, the anonymous network is considered to run on top of the Internet, *i.e.* the considered underlay network is the Internet. Anonymous communications thus take place in the *application layer* of the standard OSI protocol stack [MR10]. Since an anonymous network typically introduces *indirections*, it integrates a form of *routing*. The overall network stack consequently comprises (at least) two levels of routing: one with IP, and one in the anonymous network. As a result, for one logical hop in the anonymous network layer, *i.e.* a link between two neighboring nodes in the anonymous network's topology graph, there may be several hops in the topological graph of the IP layer. In the rest of the thesis, a *hop* designates a logical hop in the anonymous network. Furthermore, for simplicity, the routing in the underlay network is considered completely reliable (no packet loss, no interference).

The nodes' underlay identities can be considered as their IP addresses. Because an IP address can often be linked to the identity of the user owning it, it is assumed that finding one means finding the other.

Assumption 1 (Public linking of real-world and underlay identities). *Real-world and underlay identities are publicly linkable, and uncovering one means uncovering the other.*

With Internet as underlay, each node in the anonymous network can theoretically communicate with any other node in one logical hop (ignoring *middleboxes* and NAT traversal issues). However, in this work, it is assumed that the underlying topology graph is connected, but sparse (*i.e.* highly incomplete). This base assumption allows to port the results to any underlay providing a connected but incomplete graph, such as wireless mesh networks [Zha+06], or any restricted route environment [EG11]. Also, this is in accordance with some previous works in anonymous networks over Internet, that reduce the direct neighborhood of each node to a small set of other nodes in order to preserve its privacy [FM02; Cla+10].

Assumption 2 (Connected but incomplete topology graph). *The underlying topology graph is connected but incomplete.*

How nodes choose their neighbors among the overall collection of nodes in the anonymous network is a research question in itself, and out of the scope of this work. This design point is crucial, however, since a bias in a node's view of the network may lead the adversary place itself in advantageous situation. In this work, it will be assumed that the neighbor selection mechanism ensures that each client node has at least one non-adversary controlled node in its neighborhood.

Assumption 3 (Honest neighbor). *Every client node has at least one honest neighbor,* i.e. *a neighbor not controlled by the adversary.*

This work considers an open, fully distributed system, that any client node may join or leave at any time. There is no hierarchy among the nodes, and in particular, every node relays messages for its neighbors, meaning that each node uses the network to obtain privacy in its own communications, and helps other doing so as well. This is defined as the *homogeneous* architecture, as opposed to the more traditional *client-server* one that can be found in the literature.

Definition 12 (Client-Server & Homogeneous Architectures). *In a client-server architecture, client nodes only assume the role of end-senders and end-receivers, and servers are the relay nodes providing an anonymity service to the clients. In a homogeneous architecture, all nodes are simultaneously client and server, and assume the role of end-sender, end-receiver and relay.*

Note that in a homogeneous architecture, end-receivers are always part of the anonymous network, so the communications are limited to nodes inside the anonymous network

(contrarily to most client-server architectures that allow communication towards *e.g.* a plain web server that does not run specific software).

Lastly, to construct the protocol, this work assumes that there is no central server of any kind, and no trusted third party such as key servers of certificate authorities. Likewise, no *a priori* secure or private communication channels are assumed among the nodes.

1.3. Adversary Model

In the anonymous communications literature, there are several possible adversary models. These models can be described according to a combination of three criteria.

Internal/External An internal adversary takes part in the anonymous network, runs a node, and potentially acts as sender, relay or receiver. An external adversary is outside the anonymous network and can only eavesdrop communications.

Active/Passive A passive adversary can be generally described as trying not to be detected. If it is internal, it follows the protocol, if it is external, it merely observes communication links. An active adversary may deviate from the protocol, or try to replay or inject messages or tamper with messages it intercepts (even if it is external). Also, the active category of adversary includes behaviors, where a node acts in an arbitrary manner, without any particular *attack strategy* or goal. Also, an active adversary is sometimes denoted malicious, while a passive one may be called semi-honest.

Local/Global/Collusive A local adversary is restricted to a portion of the anonymous network. That is, it can only directly observe or affect a small region of the topology graph. A local *internal* adversary is a node controlled by the adversary (*i.e.* a corrupted node), while an *external* one observes a limited portion of the anonymous network (*e.g.* a couple of links). A global adversary is not limited in this sense. In particular, a global external adversary is able to observe all links and messages in the network. In between lies collusive adversaries, which can be described as a collection of two or more local adversaries sharing information and mounting coordinated attacks.

Assumption 4 (Adversary model). *The adversary is considered as a combination of* global external passive *and* collusive internal passive *adversaries working collaboratively.*

In the rest of this thesis, the adversary is considered as a powerful entity, infiltrating the anonymous network by running its own nodes or corrupting others, and recording all activities and all messages flowing through the entire network. However, corrupted nodes follow the protocol specification. This is a common model in the literature on anonymous communications, where the goal is to defend against the very network operators and governmental institutions.

The global external adversary can also be modeled by stating that all link messages going through the network are sent to it, or posted on a public *bulletin board*. This

adversary model has occasionally been deemed unrealistic in the past [Syv09]. Yet, the wide-spread eavesdropping capabilities of large entities have been demonstrated with the "Great Firewall of China" [Wal01], the FBI's carnivore system [Ste+00] and the PRISM program [Gre14].

The collusion of internal adversaries is not explicitly bounded in this work: it can grow to almost being a global adversary, as long as Assumption 3 is respected. The semi-honest model for internal adversary is quite weak, since in practice corrupted nodes are likely to cheat in order to achieve their goals. This choice of model is motivated by the fact that there is no existing methodology to systematically prove resistance against denial-of-service, byzantine or arbitrary attacks in complex communication protocols. Indeed, the ever new attacks on the Tor protocol, even after more than ten years of deployment, attests it: there may be an infinite number of ways to tamper with the protocol, and no way to check against them all. Considering passive adversaries provides a better basis for the security and privacy analysis in a first stage. In the future works section of our conclusion chapter, we put forward some modifications to the protocol that allow to resist several active attacks.

Finally, from a cryptographic perspective, the traditional probabilistic polynomial time (PPT) adversary model is employed [Gol01]. That is, the adversary has large but limited computational power. She can run algorithms which complexity is at most polynomial in the size of their inputs.

Assumption 5 (Limited computing power adversary). *The cryptographic adversary $\mathcal{A}$ is considered as a PPT Turing machine.*

1.4. Privacy Properties and Goals

This section presents the privacy properties a strongly private anonymous communication protocol should ensure. These formulations are intuitive and informal. Their formal statements are presented, in Chapter 5. Additionally, this section more generally defines the goals relative to the efficiency and functionality of the protocol, and clarifies what it *does not* aim to achieve.

1.4.1. Privacy Properties

At the highest level of abstraction, the goal is to conceal who communicates with whom, as well as the very fact that a node does communicate. That is, even though it is not possible to conceal the fact that there are communications, the protocol aims at concealing who are the end nodes of communication sessions.

To formalise these goals, the notions of *anonymity* and *unlinkability* as defined by Pfitzman and Köhntopp [PK01] are used.

Definition 13 (Anonymity [PK01]). *Anonymity of a subject means that the subject is not identifiable within a set of subjects, the* anonymity set.

Definition 14 (Unlinkability [PK01]). *Unlinkability of two or more items of interest [...] from an attacker's perspective means that within the system [...], the attacker cannot sufficiently distinguish whether these items of interest are related or not.*

The authors definition of anonymity recalls that, ultimately, an end-sender or end-receiver can only be *at best* anonymous among all the users of the anonymous network. Unlinkability is another notion extensively used in the literature on privacy. It is more versatile, allowing to define more privacy notions (actually, anonymity can be defined in terms of unlinkability [PK01]). Here, the *items of interest* are mainly the nodes' real-world identities and the end-to-end payload messages. With this terminology, five properties are defined.

Property 1 (Sender Anonymity (SA)). *The adversary can not identify the end-senders of payload messages in the network within a subset or the set of all nodes.*

Property 2 (Receiver Anonymity (RA)). *The adversary can not identify the end-receivers of payload messages in the network within a subset or the set of all nodes.*

Property 3 (Session Unlinkability (SU)). *The adversary can not correlate link messages from different communication sessions, in particular between the same end-sender and end-receiver.*

Property 4 (Message Unlinkability (MU)). *The adversary can not correlate link messages based on their* bit pattern *or from the cryptographic information they contain (or that is associated to them). In particular, she can not link messages from the same session (MU-session) nor link messages carrying the same payload message as its relayed through the network (MU-tracing).*

Property 5 (Traffic Analysis Resistance (TAR)). *The adversary can not perform traffic analysis of any form, including timing-based analysis.*

SA and RA are the main privacy goals, in the sense that they are the properties a user of the anonymous network would expect. They are standard properties, but in this work, we aim at a strong variant of them. Indeed, most work only aim at traditional notion of *relationship anonymity*, defined as the impossibility of de-anonymising both the end-sender and the end-receiver *at the same time*. Comparatively, SA alone or RA alone implies relationship anonymity. Secondly, it is important to note that, in the considered adversary model with a global network observer, SA means that it is not even possible to *observe the action of end-sending* a message (and likewise for RA and end-receiving). In the terms of Freedman and Morris, *sending activity* is not observable [FM02]. Actually, in the present adversary model, sender anonymity is arguably equivalent to sender *unobservability* (a term also defined by Pfitzman and Köhntopp [PK01]), and likewise for receiver anonymity. This definition of anonymity as unobservability can also be found, in particular, in the Tarzan protocol [FM02], and more recently in the Pung protocol [AS16].

The SU property is somewhat related to SA and RA, but not implied by them. It requires that different sessions between the same end-sender and end-receiver are not linkable, and in particular that messages exchanged between a specific pair of nodes

do not carry any distinctive mark. This would otherwise provide the adversary with material to infer information on the communicating nodes, based *e.g.* on the frequency of their exchanges. The MU and TAR properties both aim at modeling the impossibility for the adversary to gain an advantage for breaking the other properties by tracing or recognising messages. They are not equivalent, but complementary: the first is a cryptographic property, the second is a abstract, network level property. The separation is made because formally proving traffic analysis resistance is uneasy (as discussed in Chapters 3 and 5). Here, at least the MU property can be formally studied so that *cryptographically speaking*, traffic analysis can be shown impossible. Lastly, note that the distinction is made between MU-session and MU-tracing: the former requires that messages from the same communication session to be unlinkable by a given relay node, and the latter that a payload message can not be traced by two different relays on the message's path. All the above properties can be found in past literature. In particular, SU and MU are seemingly equivalent to the *no session linkage* and *no packet correlation* properties of the HORNET protocol [Che+15].

In Chapter 5, these privacy properties are reformulated using the cryptographic notion of *indistinguishability*. For instance, SA is formulated as the (near) impossibility of distinguishing between a run of the system where message m is sent to receiver R by sender S_0 from a run where m is sent to R by sender S_1.

According to the informant-journalist scenario discussed in the introduction, these properties provide strong anonymity to the informant, even against the journalist. The goal is to allow bi-directional communications between the two parties, without the journalist nor any internal or external network actor learning the identity of the informant. In this regards, SA and RA conceal the very fact that the informant communicates. More accurately, SA (resp. RA) ensures that no message can be attributed as having been end-sent (resp. end-received) by the informant, even by the journalist itself. Then, SU prevents the linking between communication sessions of the same informant and journalist. Meaning that if one session is de-anonymised, the anonymity of the others remains. TAR, MU-tracing, and MU-session prevent the tracing of the message and reconstruction of the communication pattern between the informant and the journalist, which could then lead to a breach of SA or RA (as exposed in Section 3.4, which reviews the existing threats to privacy in anonymous networks). Lastly, because the network is homogeneous and no asymmetry is introduced between nodes, the journalist actually enjoys the same privacy guarantees as the informant, with the exception that an informant initiating a communication with a journalist, of course, knows the identity of the said journalist.

1.4.2. What the Protocol Does and Does Not Achieve

Aside from the privacy goals, the protocol aims at being completely decentralised and distributed. It is above all made for non-latency sensitive communication among nodes.

However, the protocol makes no effort to prevent against denial-of-service attacks (*e.g.* refusing to relay), or to hide participation in the anonymous network. Also, making the protocol efficient is a secondary goal compared to ensuring the privacy properties.

Some essential design points of a routing protocol, such as congestion and bandwidth management are merely discussed and accounted for, but not included in the design.

1.5. Summary

This chapter laid out the foundations of this thesis, from the terminology it uses to the privacy properties it aims for. The chapter also details the context and the assumptions on the system and possible attacks under which the protocol should run. The goals in terms of privacy can be qualified as *strong*, since, as shown in Chapter 3, they are stronger than the usual properties ensured in the anonymous communication protocols literature.

2. Cryptographic Tools

The aim of this chapter is to introduce the cryptographic primitives, the *building blocks* with which the protocol is built. This chapter is rather informal from a cryptographic point of view. In particular, the formal definitions of security notions, such as the semantic security of encryption schemes, are not given here. Instead, they are deferred to Chapter 5, just before presenting the security proofs.

Before all, preliminary notions of algebra and provable security in general are very briefly recalled. Then, the concepts of public- and symmetric-key encryption are presented, followed by the description of four primitives: the SHA-3 hash function, the Diffie-Hellman key-exchange, the Elgamal homomorphic encryption scheme, and universal re-encryption. Each primitive is abstractly presented, along with its functionalities and security properties. The notion of universal re-encryption, less present in the literature, is presented in more details.

2.1. Preliminaries

As a preamble, the mathematical and cryptographic notations employed throughout the thesis are described succinctly. Then, the principle of *hard problems* and provable security in cryptography is recalled, along with the main hard problem used in this work, the Decisional Diffie-Hellman (DDH) problem.

2.1.1. Notations

Table 2.1 lists various mathematical and cryptographic symbols used throughout this thesis.

Symbol	Description	Example
$\leftarrow$	Assignment of a result to variable c	$c \leftarrow \mathsf{Enc}(pk, m)$
$:=$	Definition of a function or term	$f(x, y) := x^2$
$\leftarrow_\$$	Uniform choice of an element in a set	$x \leftarrow_\$ \mathbb{N}$
$\|$	Concatenation operator	$m_1 \| m_2$
$\cdot$	Product of two numbers or group elements (omitted when clear from context)	$x = y \cdot z$
$\|S\|$	Number of elements in a set, or bit-size of a number	$\|\mathbb{N}\|$, $\|x\|$

Table 2.1. – Mathematical and Cryptographic Notations

This work makes extensive use of groups and subgroups, and in particular *multiplicative*, *abelian*, and *cyclic* groups [Sho09]. Denoted $\mathbb{G}$, a group is hereby characterised by a generator g and an order $|\mathbb{G}|$. The product symbol "$\cdot$" is used for multiplication between group elements, and although all operations take place within a modulo n for some $n \in \mathbb{N}$, the term "mod n" is often omitted and implicit from context. The multiplication of $e_1 \in \mathbb{G}$ by e_2^{-1}, the inverse of $e_2 \in \mathbb{G}$, is sometimes noted with a division symbol e_1/e_2.

2.1.2. Cryptography and Hard Problems

Cryptographic constructions rely directly or indirectly on the assumption that some *problem* is hard to solve. For instance, the Rabin encryption scheme relies on the assumption that factoring large numbers is hard [MOV96, Section 8.3]. A problem is considered *hard* if there exists no known *efficient* algorithm that solves it, *i.e.* when it can take several years even for extremely powerful machines to solve it. Formally, a problem is *assumed* hard if it can not be solved by any known *polynomial time* algorithm, *i.e.* when there exists no PPT adversary that solves the problem. This is captured by the *security parameter* λ: a problem is hard when it takes at least $\mathcal{O}\left(2^\lambda\right)$ time to solve, *i.e.* time *exponential* in λ. The current recommendation is a security parameter of $\lambda = 128$ bits [Gir15].

In this work, the problem we are mainly interested in is the Decisional Diffie-Hellman problem. It consists in the following: for a given group $\mathbb{G} = \langle g \rangle$ and elements $g^a, g^b, g^c \in \mathbb{G}$, with $a, b \leftarrow_\$ \mathbb{Z}_q$, to distinguish whether $c = ab$ or $c \leftarrow_\$ \mathbb{Z}_q$. Intuitively, saying that the DDH problem is hard means that even if g^a and g^b are known, the term g^{ab} can not be computed, and actually *looks random*. In the rest of this thesis, "the DDH assumption" refers to the assumption on the hardness of the DDH problem.

The DDH problem is assumed hard in various groups [Bon98]. The most suitable way to instantiate $\mathbb{G}$ for this thesis, is to take a subgroup of prime order q of the multiplicative group $\mathbb{Z}_p^*$ where $p = 2q + 1$. For $\lambda = 128$, it is advised to set $|p| \approx 2048$ and $|q| \approx 200$[1].

[1] These are the values recommended by the ANSSI (the French National Cybersecurity Agency). Other organisations may recommend slightly different values [Gir15].

With a suitable generator g (such that $\exists e \in \mathbb{Z}_p^*$, $g = e^2 \bmod p \neq 1$), the group can be described as $\mathbb{G} = \langle g \rangle = \{ g^i \mod p \mid i \in \mathbb{Z}_q \}$. In the rest of the thesis, the term $\mathbb{G}$ denotes this specific group (unless stated otherwise). More details on how the group $\mathbb{G}$ can be instantiated and how elements are drawn from it can be found in Appendix A.

2.2. Public Key and Secret Key Encryption

Encryption [Gol04, Chapter 5] is the most common cryptographic primitive for ensuring confidentiality of data (or *meta-data*). Encryption schemes can be divided in two generic categories: public key encryption (PKE), and secret key encryption (SKE). These categories are also referred to as asymmetric and symmetric encryption.

The main difference between PKE and SKE is conceptual. The former uses *pairs* of keys (pk, sk) with a public and a secret (or *private*) part. Anyone can encrypt data using the *public* part of the key, producing ciphertexts that only the owner of the private key can decrypt. In SKE, there is only one key k, kept secret to typically two entities, used both to encrypt and decrypt. While PKE schemes are mainly based on number-theoretic (or, more largely, mathematical) problems, most well known SKE schemes are based on block or stream ciphers. It is known that SKE schemes are much more efficient (for the same security level λ), but less flexible than PKE schemes. In particular, SKE requires the communicating parties to share a common symmetric key before any communication can take place (by agreeing, or exchanging one), while the public key in a PKE scheme allows one to straight away encrypt and send data to other parties. As a result, it is common to perform encryption in a *hybrid* way: to take advantage of the efficiency of SKE, the data is encrypted with a symmetric key k, and the latter is sent encrypted under the public key of the recipient.

This work uses PKE extensively, and SKE in specific occasions. Below is a generic description of a PKE scheme, under its *probabilistic* form.

Definition 15 (PKE scheme). *Given a plaintext space $\mathcal{P}$, a ciphertext space $\mathcal{C}$, a key space $\mathcal{K} = (\mathcal{K}_{pk} \times \mathcal{K}_{sk})$, and a random coins space $\mathcal{R}$, a probabilistic PKE scheme consists of (at least) the following three operations:*

Key Generation: $\mathsf{KeyGen}(1^\lambda) : \{0,1\}^* \to (\mathcal{K}_{pk} \times \mathcal{K}_{sk})$
A probabilistic polynomial time algorithm outputting a key pair (pk, sk) achieving the level of security specified by the security parameter λ, where pk denotes the public key *whilst sk denotes the* private key*.*

Encryption: $\mathsf{Enc}(pk, m, r) : \mathcal{K}_{pk} \times \mathcal{P} \times \mathcal{R} \to \mathcal{C}$
A deterministic polynomial time algorithm that, given a public key pk and a plaintext m, outputs a ciphertext c encrypting m with the randomness r.
Alternatively: $\mathsf{Enc}(pk, m) : \mathcal{K}_{pk} \times \mathcal{P} \to \mathcal{C}$ can be described as a probabilistic algorithm, where r is internally and randomly sampled.

Decryption: $\mathsf{Dec}(sk, c) : \mathcal{K}_{sk} \times \mathcal{C} \to \mathcal{P}$
A deterministic polynomial time algorithm that, given a private key sk and a

ciphertext c, outputs m if $c \leftarrow \mathsf{Enc}(pk, m)$ and sk is the private key corresponding to pk.

A SKE scheme also roughly follows the same description, except that it only handles one *secret key* k. For short, symmetric encryption of plaintext m with k is denoted $\{m\}_k$. Additionally, the randomness r used by a SKE scheme is called initialisation vector (IV) and, contrarily to a PKE scheme, can safely be made public and sent along with the ciphertext it relates to.

A PKE (or SKE) scheme achieves semantic security, a notion also known as indistinguishability under chosen plaintext attacks (IND-CPA), if, given a ciphertext, the adversary can not learn anything about the underlying plaintext. More formally, this is captured by the impossibility for the adversary to distinguish whether a ciphertext c is an encryption of m_0 or m_1 when m_0 and m_1 are known and chosen by the adversary herself. IND-CPA is the notion of security used in this work. However, there exists stronger notions, such as indistinguishability under chosen ciphertext attacks (IND-CCA) security, where the adversary is given additional capacities. Namely, it is given the opportunity to decrypt any ciphertexts she wants except of course the challenge ciphertext c, as this would immediately tell her if it is an encryption of m_0 or of m_1.

2.3. Cryptographic Hash Functions

A hash function $\mathsf{h}(x) : \{0,1\}^* \to \{0,1\}^n$ is a deterministic function efficiently mapping an input x of arbitrary length to an output of fixed n-bit length. A *cryptographic hash function* exhibits additional properties, such as the well known *preimage resistance*, *2nd-preimage resistance*, and *collision resistance* properties [MOV96; RS04], that (informally) prevent from inverting the function, or finding two inputs that hash to the same value (the latter is called a *collision*).

Hash function are commonly used to ensure *data integrity* of messages sent over untrusted communication channels. But they can also be used to design key derivation functions (KDF) (to derive keys from a shared secret), or pseudo-random functions (PRF) (to produce unpredictable sequences of bits) [MOV96].

In this thesis, the SHA-3 hash function [NIS14] is used as a KDF to derive many keys from a single secret, and as a PRF to transform algebraically related inputs into (seemingly) unrelated data. SHA-3 is based on the *Keccak* function [Ber+11], which realises preimage resistance, 2nd-preimage resistance and collision resistance. More accurately, the security of *Keccak* is stated in comparison to a truly random function, which implies that it realises all three properties, and makes it suitable to be used as KDF and/or PRF.

2.4. Key Agreement

The most used method for generating common secrets, in particular over the Internet, is the Diffie-Hellman key agreement (DHKA), or *Diffie-Hellman Handshake* [DH76]. It

typically involves two parties X and Y, that exchange elements from a group in which the DDH assumption holds. With the group $\mathbb{G}$ defined in Section 2.1.2, the protocol is executed as follows. X sends $A = g^a \in \mathbb{G}$ for $a \leftarrow\!\!\$\, \mathbb{Z}_q$, then Y answers with $B = g^b \in \mathbb{G}$ for $b \leftarrow\!\!\$\, \mathbb{Z}_q$, and they finally compute $secret = B^a = A^b \in \mathbb{G}$. Since the DDH assumption holds in $\mathbb{G}$, it is ensured that only X and Y know the secret, and that it constitutes a suitable random seed that can be fed *e.g.* to a KDF or PRF.

In this thesis, the DHKA and a KDF are used to generate many secrets shared by two neighboring nodes. It is known that this basic, unauthenticated version of the DHKA is subject to a *man-in-the-middle* attack. However, this attack is not part of the *passive* adversary model considered in this work. Furthermore, the use of an authenticated version of the DHKA requires *e.g.* to assume that parties possess public keys certified by a trusted authority [Gol04]. However, in this thesis, we aim at avoiding the reliance on such a central entity, and prefer a fully distributed architecture.

2.5. Homomorphic Encryption (HE)

Traditional encryption transforms a plaintext into a random-looking bit-string which can not be of any use to anyone without the corresponding decryption key. A homomorphic encryption (HE) scheme differs in that it allows one to apply transformations to a ciphertext, which map to known and predictable transformations on the underlying plaintext, without leaking information on the latter. That is, in a HE scheme, there exists a *homomorphism* from the ciphertext space $\mathcal{C}$ to the plaintext space $\mathcal{P}$. For instance, in some HE scheme such as Paillier's, the *multiplication* of two ciphertexts $\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2)$ results in a ciphertext $\mathsf{Enc}(m_1 + m_2)$ encrypting the *addition* of their plaintexts [Pai99]. The applications of HE include electronic voting, private information retrieval, secure multi-party computation, and more generally, the protection of privacy in cryptographic protocols [Rap06]. The most direct application, however, is the secure delegation of computation, without the need to reveal the data on which the computation is performed. For instance, it is possible for a device with low computation power to delegate heavy computations to the cloud. For that, the device encrypts its data with a HE scheme, sends it to the cloud, which performs the computations and send the result back. The cloud never learns details about the data it processed, only its *nature* (*e.g.* it knows how it is encoded, and has the knowledge of how to handle it).

Initially described by Rivest *et al.* [RAD78], HE subsequently attracted a lot of a attention. A major milestone was passed with the first *fully homomorphic encryption* scheme proposed by Gentry in 2009 [Gen09], which is capable of evaluating *any* function (that would be normally computable on clear data) on encrypted data. However, even though great advances have been made in the recent years, fully HE remains impractical as of today. In this thesis, we resort to simple, less powerful schemes that only allow a restricted set of operations.

The flexibility given by HE comes at the cost of reduced security. Indeed, HE schemes are at best IND-CPA secure, and by definition can not achieve IND-CCA security. This is mainly due to the *malleability* of the ciphertexts, and to the fact that learning the

decryption of one ciphertext $\mathsf{Enc}(m)$ means getting information about all other ciphertexts that are known to encrypt a function of the plaintext m. In practice, if this downgrade in security poses a serious threat, one can use authentication of plaintexts or ciphertexts via integrity-checking tools, such as message authentication codes, or another layer of traditional encryption on top of the HE scheme.

The Elgamal HE scheme

In this work, HE is used in particular to privately compute anonymous network identities. Because its homomorphic properties are adapted to our needs, we use the Elgamal PKE scheme [Elg85]. This scheme works over any group in which the DDH assumption holds, but it is presented here for the specific group $\mathbb{G}$ considered in this work. The presentation below features the KeyGen, Enc, Dec operations proper to PKE schemes.

- $\mathsf{KeyGen}(1^\lambda)$: Given $\mathbb{G}$ described by g and q, pick a random $x \in \mathbb{Z}_q$ and compute $h = g^x \in \mathbb{G}$. Output $(pk, sk) = (h, x)$.
- $\mathsf{Enc}(pk, m, r)$: For $m \in \mathbb{G}$ and a random $r \in \mathbb{Z}_q$, output $c = (g^r, m \cdot h^r) \in \mathbb{G}^2$.
- $\mathsf{Dec}(sk, c)$: Let $c = (c_0, c_1)$. Compute and output

$$\frac{c_1}{c_0^x} = m\frac{h^r}{(g^r)^x} = m\frac{g^{xr}}{g^{xr}} = m \bmod p$$

The Elgamal scheme is semantically secure under the assumption that the DDH is intractable in the group $\mathbb{G}$. Under the same assumption, the Elgamal additionally satisfies the key-privacy property, ensuring that it is impossible for the adversary to distinguish which key among two or more candidate keys was used to encrypt some ciphertext. This notion is also called indistinguishability of keys under chosen plaintext attacks (IK-CPA) [Bel+01].

The scheme allows the following homomorphic operations[2], for $m, m' \in \mathbb{G}$, ciphertexts $c = (c_0, c_1) = \mathsf{Enc}(pk, m, r)$ and $c' = (c'_0, c'_1) = \mathsf{Enc}(pk, m', r')$, and $\alpha \in [0, |\mathbb{G}| - 1]$:

(1) Multiplication: multiplication of the plaintexts underlying two ciphertexts

$$\begin{aligned}\mathsf{CtxtMult}(c, c') &\coloneqq (c_0 \cdot c'_0, c_1 \cdot c'_1) \\ &= (g^{r+r'}, m \cdot m' \cdot h^{r+r'}) \\ &= \mathsf{Enc}(pk, m \cdot m', r + r')\end{aligned}$$

(2) Plaintext multiplication: multiplying an underlying plaintext by an other plaintext

$$\begin{aligned}\mathsf{PlainMult}(c, m') &\coloneqq (c_0, c_1 \cdot m') \\ &= (g^r, m \cdot m' \cdot h^r) \\ &= \mathsf{Enc}(pk, m \cdot m', r)\end{aligned}$$

[2]Other operations are possible, but here are listed only the relevant ones for this thesis.

(3) Scalar exponentiation: exponentiating an underlying plaintext

$$\begin{aligned}\mathsf{ScExp}(c, \alpha) &\coloneqq (c_0^\alpha, c_1^\alpha) \\ &= (g^{r\alpha}, m^\alpha \cdot h^{r\alpha}) \\ &= \mathsf{Enc}(pk, m^\alpha, r\alpha)\end{aligned}$$

(4) Key Homomorphism: for $(pk', sk') = (h', x')$

$$\begin{aligned}\mathsf{KeyMult}(sk', c) &\coloneqq (c_0, c_1 \cdot c_0^{x'}) \\ &= (g^r, m \cdot (h \cdot h')^r) \\ &= \mathsf{Enc}(pk \cdot pk', m, r)\end{aligned}$$

The inverse operation is simply $\mathsf{KeyDiv}(sk', c) \coloneqq (c_0, \mathsf{Dec}(sk', c))$. For short, the operation Dec is used to denote KeyDiv throughout the thesis, since the latter can actually be seen as a *partial decryption*.

Notice how the last operation, KeyMult, actually shows that the Elgamal scheme supports encryption under multiple keys. That is, a plaintext m can be encrypted as $c \leftarrow \mathsf{Enc}(pk_1 \cdot pk_2 \cdot \cdots \cdot pk_n, m)$. It can either be decrypted in one sitting, with $\mathsf{Dec}(\sum_i sk_i, c)$, or in multiple steps using $\mathsf{Dec}(sk_i, c)$. Since $\mathbb{G}$ is abelian, the order of keys do not need to be respected when decrypting in multiple steps (*e.g.* $\mathsf{Dec}(sk_5, \mathsf{Dec}(sk_3, c))$ is the same as $\mathsf{Dec}(sk_3, \mathsf{Dec}(sk_5, c))$).

2.6. Universal Re-encryption (URE)

This work makes heavy use of re-encryption and universal re-encryption as a means to modify the appearance of ciphertexts, and ultimately prevent the tracing of messages in the network. This section first presents standard re-encryption, and then the notion of universal re-encryption.

2.6.1. Re-Encryption

Re-encryption, in a probabilistic PKE scheme, consists in changing the random coins r embedded in a ciphertext $c = \mathsf{Enc}(pk, m, r)$, while leaving the plaintext m untouched. Indeed, recall that in a probabilistic PKE scheme, for the same public key, a single plaintext m has many possible different encryptions depending on the value of r. The motivation behind re-encryption is changing the appearance of a ciphertext in such a way that it is unrecognisable, even given information on the original ciphertext.

The Elgamal scheme, as most HE scheme, supports re-encryption. The re-encryption of ciphertext $c = \mathsf{Enc}(pk, m, r) = (c_0, c_1) = (g^r, m \cdot h^r)$ into c' is performed, given $pk = h$, by sampling $r' \leftarrow_\$ \mathbb{Z}_q$ and computing:

$$\begin{aligned}c' &= (c_0 \cdot g^{r'}, c_1 \cdot h^{r'}) \\ &= (g^{r+r'}, m \cdot h^{r+r'}) \\ &= \mathsf{Enc}(pk, m, r + r')\end{aligned}$$

Intuitively c' is unrecognisable because r' is uniformly random, thus $g^{r'}$ and $h^{r'}$ are uniformly random as well and act as a *mask* for c_0 and c_1 respectively. Note that the plaintext m does not need to be known to the entity carrying out the re-encryption.

In anonymous networking, the main application of re-encryption is to modify the appearance of messages as they travel through an anonymous network, to prevent their tracing[3]. It is actually one of the alternatives to the more common technique of *onion routing* (see Chapter 3). However, to re-encrypt messages transiting in the network, the knowledge of $pk = h$ is necessary, in order to multiply the second component of c with $h^{r'}$. This can be an issue, because a public key can act as a global identifier in the anonymous network, which we want to avoid in this work (see Chapter 4 for details). The use of public keys in re-encryption can be avoided by resorting to universal re-encryption (URE).

2.6.2. Universal Re-Encryption

The notion of URE was proposed in 2004 by Golle *et al.* [Gol+04], along with an example of URE-enabled scheme. A URE-enabled scheme exhibits a UReEnc operation in addition to KeyGen, Enc, and Dec. This operation does not necessitate nor leak information on the public key of re-encrypted ciphertexts. The authors also propose a new notion of security, suitable for schemes supporting URE, named universal semantic security under re-encryption (USS). It is based on the traditional notion of semantic security of PKE schemes, and additionally requires that re-encrypted ciphertexts are unrecognisable from their original ciphertext. More precisely, USS states that an adversary knowing $pk_0, pk_1, m_0, m_1, r_0$, and r_1 should not be able to distinguish $\mathsf{UReEnc}(\mathsf{Enc}(pk_0, m_0, r_0))$ from $\mathsf{UReEnc}(\mathsf{Enc}(pk_1, m_1, r_1))$.

In the same work, Golle *et al.* present an extension of the Elgamal scheme supporting the UReEnc operation, hereby called *URE-Elgamal*. In this scheme, a plaintext is encrypted as a pair of Elgamal ciphertexts: the first one encrypts the plaintext, and the second is an encryption of the identity element of the group $\mathbb{G}$ (*i.e.* an encryption of one). An encryption of one c_{one} in the (standard) Elgamal scheme has the following properties: (i) anyone can generate new re-encryptions of the ciphertext c_{one} without knowledge of the public key that encrypted it; and (ii) given solely an encryption of one, anyone can re-encrypt any ciphertext using the homomorphic properties of the scheme. These two properties thus allow the re-encryption of any ciphertext, without public keys.

More precisely, the UReEnc operation of Golle *et al.* is defined as follows, on input $C = (\mathsf{Enc}(pk, m, r), \mathsf{Enc}(pk, 1, r_{\text{one}})) = ((c_0, c_1), (c_{\text{one}0}, c_{\text{one}1}))$:

$$\begin{aligned}\mathsf{UReEnc}(C) &\coloneqq \Big((c_0 \cdot c_{\text{one}0}^{\,s}, c_1 \cdot c_{\text{one}1}^{\,s}), (c_{\text{one}0}^{\,s'}, c_{\text{one}1}^{\,s'})\Big) \text{ with } (s, s') \leftarrow_{\$} \mathbb{Z}_q^2 \\ &= \big(\mathsf{Enc}(pk, m, r + r_{\text{one}} \cdot s), \mathsf{Enc}(pk, 1, r_{\text{one}} \cdot s')\big)\end{aligned}$$

The authors prove the USS property of the URE-Elgamal scheme, based on the IND-CPA and IK-CPA property of the plain Elgamal scheme. The drawback of the

[3] Jakobsson however uses it to build a *verifiable mixnet*, *i.e.* a network that detects malicious behavior from relay servers [Jak99].

construction, however, is its inefficiency. Indeed, a plaintext of n bits becomes $4n$ bits of ciphertext (against only $2n$ bits for plain Elgamal), and re-encryption requires 4 modular exponentiations and two multiplications.

Several works make use of the URE-Elgamal scheme, to change the appearance of messages and prevent their tracing [Gol+04; GKK04; Lu+05; HLF12]. The advantage of URE over standard re-encryption in this regards, is that it does not require a (public) key distribution at the initialisation of the network. The first anonymous network using URE is from Golle *et al.* (in the same work), who propose a straightforward application of the scheme to construct a protocol in the *bulletin board* model (a public structure where any party can read or write), yielding a rather theoretical protocol. The same year, Gomułkiewicz *et al.* [GKK04] (surprisingly) used URE to build an onion routing protocol. Note that the privacy guarantees of the works of Gomułkiewicz *et al.* [GKK04] and Lu *et al.* [Lu+05] have actually been broken by Danezis [Dan06], but the attacks exhibited do not pertain to URE itself but rather to a misuse of the technology. More recently, URE has been used in an anonymous network construction by Huang *et al.* [HLF12]. The authors actually make use of standard and universal re-encryption alternatively. The former for ciphertexts encrypted under the public keys of relay servers of the network, and the latter for ciphertexts encrypted under the receiver's public key pk_R, thus preventing the relays from learning pk_R, and ultimately deducing the identity of the receiver.

In this thesis, it is the latter approach that is chosen: all messages are encrypted under the receiver's public key, URE is used to change their appearance, and relay nodes do not learn the receiver's public key nor identity. In practice, the UReEnc operation is broken down into several functions, which are used individually on a need-basis. Indeed, by the way the protocol is designed, full URE-Elgamal ciphertexts are not always necessary: in many occasions, messages do not need to embed an encryption of one, as nodes will already have a suitable one available. As a result, bandwidth is saved, and the workload for re-encryption is reduced. The atomic functions are described in detail in Chapter 4. In their use, care is taken to reproduce the UReEnc operation so that the USS security defined by of Golle *et al.* [Gol+04] still holds.

2.7. Summary of Cryptographic Tools

In this chapter, the four main cryptographic primitives that will be used to design the protocol have been presented, recalling necessary mathematical and cryptographic notions beforehand.

The main assumption on which the security of the primitives rely is the hardness of the DDH problem, since the DHKA, the Elgamal scheme, and the URE constructions all rely on it. Also, all three primitives work in the same group $\mathbb{G}$. As a matter of fact, most of the operations and elements constitutive of the protocol will lie in $\mathbb{G}$.

3. Background and Related Works

This chapter presents the state of the art in privacy-preserving communication protocols, using the terminology introduced in Chapter 1. The aim is, ultimately, to show how the work in this thesis builds on pre-existing ones, and to review the potential threats against the SA, RA, SU, MU and TA privacy properties in anonymous networking.

Several sub-domains in privacy-preserving communication protocols co-exist: *direct messaging* [Cha+16], *file sharing* [Cla+01], those focused on *electronic mail* [Mol+03], the *electronic voting* protocols [MN10], and the *multi-purposes* protocols [DMS04]. The focus in this thesis is on *direct messaging* protocols, that allow any two entities to directly communicate any kind of data over the Internet. Consequently, and to narrow the scope of this survey, only the protocols having this explicit goal are presented.

Then, the literature on privacy-preserving direct messaging protocols distinguishes two general categories, based on the latency they introduce in the delivery of messages:

high and low latency protocols. These categories are sometimes respectively identified to mixnet and onion routing [Cha+16]. However, there are many exceptions to this categorisation. For instance, some onion routing protocols introduce high latency [GT96; DDM03; Mol+03; SSH08], which can be confusing. Hereby, a categorisation in three classes is proposed. The first consists in low latency systems (not necessarily associated with onion routing), which includes the well known Tor protocol [DMS04]. The second is focused on high latency systems, and in particular on the mixnets. The last category relates to (high or low latency) homogeneous networks (as defined in Chapter 1), which is of particular relevance since the protocol proposed in this work considers a homogeneous architecture.

These three categories however share a common base idea for providing anonymity. It is better explained by coming back to the most basic and simple way to obtain anonymity on the Internet: using a *proxy*. A proxy is an intermediary server that makes the request on behalf of a client, thus concealing its identity. However, using only one proxy means completely relying on its honesty. The proxy knows the sender and receiver (*e.g.* the client and the web server requested), and may keep records of past requests or divulge them. Thus, it is common to use *several* such intermediaries. This is the most common approach to anonymous networking, and the base idea of all anonymous networks presented in this chapter.

The chapter begins with three sections, one for each identified protocol category. After a description of the category, a representative protocol is described, along with its claimed anonymity properties. Then, Section 3.4 reviews existing attacks on anonymous networks, that directly or indirectly participate in breaching privacy (mainly by finding senders or receivers). As attacks are described, their impact on each protocol category is studied. This survey of attacks aims at highlighting the necessary safeguards and mechanisms to ensure SA, RA, SU, MU and TAR. The last section concludes and places the present work in the continuation of existing ones.

3.1. Low Latency Networks

The most efficient and practical privacy-preserving protocols on the Internet are low latency ones [BG03; DMS04; Che+15; AS16; I2P]. Most low latency networks seek to realise *relationship anonymity*, stating that it is impossible uncover the communication relations, *i.e.* de-anonymise end-senders and the end-receivers *at the same time*. The specificity of low latency network, is above all to aim at introducing these indirections with minimal overhead compared to a direct sender-receiver communication. That is, the trade-off between efficiency and privacy is tilted in favor of efficiency here. As a result, low latency protocols support time-sensitive applications such as live chats and web browsing. This explains the popularity of such protocols over high latency ones among the general public. For instance, the Tor [DMS04] and I2P [I2P] protocols are deployed over the Internet and fully operational.

3.1.1. Building Blocks and Properties of Low Latency Networks

Low latency networks are usually based on the client-server architecture. As such, network edges are *observable*: SA and RA do not hold. Indeed, the first relay server (or an external observer of the link between it and the sender) detects the sending activity of S, and breaks SA as defined in Section 1.4.1. The same applies to the last relay server and RA. On the other hand, the client-server architecture puts little burden on users, and allows them to contact receivers outside of the anonymous network (*e.g.* a regular web server).

Note that even though none of SA nor RA holds, relationship anonymity may still hold. It is only necessary to have (at least) one honest relay, and to prevent non-neighboring relays from recognising messages as they travel down the relay servers. More generally, the tracing of messages must be prevented. However, the specificity of low latency protocols is to prevent traffic analysis only *up to a point*, as long as it does not impact efficiency *too much*. In view of the MU/TAR separation made in Section 1.4.1, this class of protocols only aims at a form of MU. In particular, most protocols encrypt messages, and change their appearance at each hop. This last point is usually performed either by decrypting and re-encrypting at each relay server [Cha81], using onion encryption [DDM03], using (universal) re-encryption [Gol+04], or simply using random bit-strings as masks [DG09]. In addition, some works ensure that all link messages are of the same size, to prevent tracing based on size.

However, network-level traffic analysis (corresponding to TAR), based on timing or traffic shape for instance, is usually not prevented at all. The rationale being that the cost of integrating such protection is either prohibitive for the user experience, or simply too costly compared to the security guarantees it brings. As a result, in most low latency protocols, corrupting the first and last relays of a communication is enough to completely break anonymity, and uncover which end-sender and end-receiver communicate together. This may even be possible without corrupting end relays, but merely by *observing* the first link (between sender and first relay) and the last link (between last relay and receiver) of the communication.

Finally, note that low latency networks do usually (implicitly) ensure a form of SU (or variants of it), in order to realise relationship anonymity. The usual adversary model of low latency networks is weaker than the one considered in the present work, mostly in that the external adversary is assumed local or collusive but not global. Also, they achieve their anonymity goals only under the assumption that no traffic analysis is possible, or equivalently, assuming that the first and last relays (and links) are not corrupted.

3.1.2. Description of Tor

As a study case, we present the Tor protocol [DMS04] and its claimed anonymity. It is the most successful anonymous protocol to date, serving today more than two millions users. Tor runs over the Internet without restrictions on the topology graph (any Tor node can directly contact any other), and based on a client-server architecture. Server nodes are called *onion routers* (OR). Anyone may freely run an OR, or join the network as a user. Users rely on existing ORs to create a circuit and tunnel their connections to

a receiver out of the anonymous network, such as a web server. In practice, a circuit is always made of three ORs chosen by the user. They are called the *entry*, the *middle*, and the *exit* nodes. It is assumed that the sender knows the ORs and their certified public key. For that, Tor provides *directory servers*, a small set of trusted servers responsible for publishing information on the network to all nodes. The circuit construction by the sender consists in distributing the *circuit identifiers* to the chosen ORs. The circuit construction is *telescopic*, and depicted in Fig. 3.1. For compactness, a case with only two ORs is shown. Circuit construction with a third OR is easily deduced. Communications between pairs of nodes are assumed to work over a secure TLS connection to counter external adversaries. In Fig 3.1, Enc designates RSA encryption, $\{m\}_k$ designates AES encryption of message m with key k, and h the SHA-1 hash function.

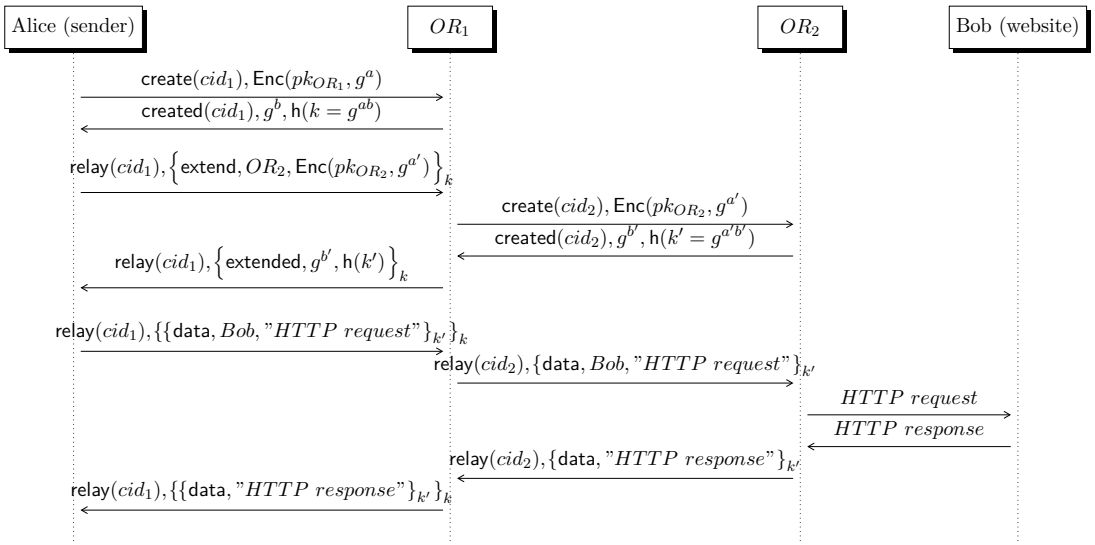

Figure 3.1. – Circuit Construction in Tor (inspired from [DMS04, Fig. 1])

For a circuit over two ORs, the construction takes two steps. First, the sender and OR_1 exchange create and created routing messages effectively realising a DHKA to obtain an AES key k. This key agreement is authenticated from the point of view of the sender (*i.e.* it can check that the key is shared with the OR it expects), since the first half is encrypted with OR_1's certified public key, and the value of the key is confirmed by h(k). In this first phase, the sender also communicates the circuit identifier cid_1 it chose to OR_1.

In the second phase, another DHKA is carried out between the sender and OR_2 to agree on a key k', with OR_1 acting as relay. For that, the sender gives a relay message with cid_1 to OR_1, and a nested extend message encrypted with k (which OR_1 can decrypt). The extend message instructs OR_1 to send a create message to OR_2, with a circuit identifier cid_2 of its choice, identifying the circuit link between OR_1 and OR_2. cid_2 is completely independent of the value of cid_1. OR_2 answers to OR_1 as if the latter was the original sender, and OR_1 forwards its answer back to Alice in an extended message nested within a

relay message. Note that OR_1 can not disturb the DHKA or perform a man-in-the-middle attack on the DHKA, without Alice (the sender) noticing it.

Once the circuit is built, the sender can start sending data to *e.g.* a web server in this example. For that, the sender creates a relay message for cid_1, accompanied with a doubly encrypted data packet: the outer layer of encryption is with k, and the innermost with k'. OR_1 handles the message as follows: by cid_1, OR_1 knows that it must use k to decrypt the nested message, replace, cid_1 by cid_2, and forward to OR_2. Each OR on the route acts in the same way, and the last, OR_2 in this example, gets the data command, instructing to send a HTTP request to the server named Bob. More generally, any IP packet can be tunneled through Tor, not only HTTP ones. When OR_2 gets the HTTP response, it sends it back encrypted under k' to OR_1 with cid_2. OR_1 re-encrypts it with k, and forwards it to the sender, which can decrypt the two layers of encryption and get the HTTP response.

There is a possibility in Tor to communicate with receivers inside or outside the network, without even knowing its IP address. This mechanism is called *hidden services*, and allows a receiver to be contacted *via* an anonymous address, not publicly linked to its real-world identity. For that, a receiver builds a reverse circuit using ORs it choses, and keeps it alive. In that circuit, the OR farthest to the receiver is its *rendez-vous point*. Its IP address is published in directory servers, along with the receiver's anonymous address. When a sender wants to contact that receiver, it queries the directory servers, gets the *rendez-vous* point, and creates a circuit towards it. The sender then starts sending message containing a specific command and the receiver's anonymous address, essentially instructing the *rendez-vous* point to forward those message into the circuit built by the receiver. Hidden services are thus implemented by joining two circuits, one from the sender, and one from the receiver.

Finally, the protocol uses fixed length messages, provides ways to manage, open, or destroy circuits, and to manage message flows.

The *nested* encryptions structure in the above description is generally called an *onion*, and protocols based on onions are called onion routing protocols. Onion routing is a way, as URE, to modify the appearance of messages at each hop in a cryptographically robust manner. Generally, the onion is created by the sender, and a layer is peeled off by each of the nodes on the route. The Tor onion structure has the particularity of not growing in size with the number of hops the message makes. Indeed, data is encrypted *in place*, and since with AES, a plaintext of n bits produces a ciphertext of n bits, encrypting a plaintext multiple times always results in n bits[1]. However, many onion routing protocols exhibit onions structures that grow in size, such as in Mixminion [DDM03], because at each layer, they embed routing information for each relay server on the route.

Tor solely aims at relationship anonymity. It exhibits the inherent vulnerabilities of low latency networks. As the authors note since the early design stages of the protocol [DMS04,

[1] IVs are not sent along the ciphertexts: Tor actually uses AES in counter-mode, always initialising the IV to zero.

Section 7], corrupting entry and exit relays, or observing the edges of the network allows to re-link senders and receivers.

Additionally, various elements of design in the protocol degrade privacy to increase performances. In particular, the OR selection made by users is biased towards nodes with most bandwidth. This allows a powerful attacker with large resources to place itself at strategic points in circuits. In the same idea, directory servers constitute targets of choice for an adversary willing to advertise false information on ORs. Alternatively, simply observing which user queries which directory server can help track users based on their assumed view of the network [DDM03, Section 7].

3.1.3. Concluding on Low Latency Networks

Low latency protocols are well suited for a use by a broad public, and to provide *anonymity for the masses* [Lin16b]. Indeed, although it provides somewhat *weak* anonymity, it is sufficient for regular web users simply concerned for their privacy in their everyday use of Internet. The protections put in place are not extremely robust, but are enough to dissuade the adversary in investing resources to mount an attack against such targets.

In view of the privacy features we target in this work, however, this level of anonymity is not sufficient. First, because network edges are observable, and secondly because traffic analysis (in particular, possible even for external adversaries) allows to re-link end-sender and end-receiver activities, and completely breach privacy.

3.2. High Latency Networks and Mixnets

Historically proposed prior to low-latency systems, mix networks, or mixnets [GT96; DDM03; Mol+03; SSH08; DG09; Hoo+15; Kwo+15; Cha+16] were introduced by Chaum in 1981 [Cha81]. This seminal work has inspired a long line of contributions in all types of anonymous networking. Mixnets also rely on the client-server architecture, but their main difference with low latency protocols is that mixnets aim, by design, at preventing the TAR version of traffic analysis. For that, each relay server, called a mix, in addition to changing the appearance of messages, carefully re-orders them and/or adds random delays during the forwarding. Some mixnets also make use of dummy messages, fake messages with random payloads that are indistinguishable from actual messages. The idea with dummy messages is to introduce *noise* and perturb the adversary's traffic analysis.

Usually, mixnets aim for relationship anonymity, against a global eavesdropper or a collusion of nodes. SA and RA are usually not achieved by mixnets, due to the client-server architecture that allows observation of network edges, *i.e.* the first relay or an observer of the first link breaks SA (and likewise for RA). Mixnets however do ensure MU and TAR both, and usually realise a form of SU as well.

Re-ordering and delaying messages to ensure TAR comes with a great cost in latency. Mixnet thus do not support applications such as web browsing, SSH connections or more generally any applicative protocol sensitive to gaps in message flows. Consequently, the main application for mixnets are email communications (the so-called *remailer* [DDM03]).

Such mixnets have been deployed and are still active today. For instance, the Mixmaster and Mixminion protocols run between 20 and 30 mix nodes, and are still used on a daily basis [Moc06].

Before delving into the detailed description of mixnets and their characteristics, note that those do not represent the only type of high latency network. In particular, a approach common to some recent works [CBM15; AS16] consists in having senders drop their messages on a central (not necessarily trusted) server. Receivers must then retrieve the messages that are meant to them. To conceal which sender communicates with which receiver, this latter task can be done by having the server broadcast all messages to all receivers, who can then select the ones that are meant to them. Another solution is to use *private information retrieval*, a generic cryptographic primitive that (in this application) allows receivers to retrieve messages from the server, without the latter actually learning which ones. These approaches get around the issue of traffic analysis, since there is no actual flow of messages. However, the remainder of this section solely focuses on mixnet. Firstly, because the described approaches only works with a client-server architecture, which we do not use, and also because this thesis adopts several techniques proper to mixnets.

3.2.1. The Different Types of Mixnets and Their Properties

There are several to many ways to implement a mixnet. To characterise a mixnet, there are two main criteria: the way each single mix in the network functions, and the way the mixes are arranged together.

Different Types of Mixes Globally, the internal functioning of a single mix depends on: (i) the way it re-orders messages, (ii) whether it produces *dummy messages* or not and how, and (iii) on the way it changes the appearance of messages. The first criterion is the most crucial. We distinguish between batched and continuous mixes. The former type of mix retains the messages it receives until its *firing condition* is fulfilled (*e.g.* until it has at least a certain number of messages, or until some time-based condition is fulfilled), and then sends them all or a portion of them in a random order. The interval between two *firing* of the batched mix is called a *round*. In periods of low traffic, this means that a message may be retained at a mix for a few hours, and up to twenty four hours or more [MD05]. To avoid these situations, continuous mixes simply retain messages for a random delay *independent* for each message. There are several variants of batched mixes [SDS02] and continuous mixes [Dan04]. Batched mixes mainly differ by their *firing condition*. Among them are pool mixes, with complex firing conditions, and which do not always forward all the messages received in a previous time frame. Continuous mixes mainly differ by the random distribution used to sample the messages' delays. Note that, in a given mixnet, all mix nodes are of the same type, and follow the same behavior. A mixnet can not be a heterogeneous collection of several types. By extension, the mixnet itself is thus said batched or continuous. The second criterion distinguishes mix nodes based on dummy messages. Dummy messages can either be sent as a link message between two neighboring mixes, or as an end-to-end message (*e.g.* from the

first to the last mix on the route). The first ones can be used to conceal the number of actual link messages exchanged between two mixes from an external observer, while the latter one can be used to thwart end-to-end traffic analysis of sessions flows. Each policy of dummy messages puts a burden on the network, and consumes bandwidth. Finally, in all mixnet implementation, mixes pad messages to a constant size, and change their appearance, roughly in the same manner as in low latency networks (usually, using some cryptographic primitive). Combining all these three criteria, many variants of mixes were proposed along the years, most of which are referenced in a taxonomy by Dias and Preneel written in 2004 [DP04], only slightly outdated.

Different Network Organisations Given a specific implementation of a mix, there are then several ways to combine mixes together, and thus several topological organisations for mixnets [DSS04]. The most common and widely studied is the cascade, where all mixes form a single line, and users have no choice but choosing the first mix of the line as their first relay. In this setting, it is known that, as long as one of the mix server in the cascade is honest, senders and receivers can not be linked together. A second type of network, called stratified, consists in several cascades. This offers more flexibility to the users, since they can choose between different cascades, based for instance on their trust in the mixes composing each of them. Finally, the highest flexibility is offered by free-route mixnets, where user may, as in Tor, choose freely the sequence of mix servers for their messages. Free-route mixnets are also the least studied, because more complex to formalise than simple cascades.

Secondly, a mixnet can function in a synchronous or asynchronous manner. In the former case, which mainly applies to networks with batched mixes, all the mixes in the network are synchronised and are at any time in the same *round* (*i.e.* they *fire* and send messages all at the same time). In the latter case, each mix functions independently, may process its messages as soon as received and *fire* independently from the other mixes. Note that a cascade mixnet is, by definition, synchronous. Synchronicity mainly has an impact on the security (or privacy) of free-route mixnets: although it is agreed that asynchronous free-route networks are not secure, synchronous ones are deemed acceptable [DSS04]. The rationale being that in asynchronous free-route networks, each mix node processes a heterogeneous collection of messages that are different distances away from their receiver and sender. And this may give away information on the length of the route, and on senders and receivers.

Overall, a thorough comparison of all solutions and a clear statement of the consequences of each design choice is still needed, and would require a substantial effort from the whole community. In particular, there is no existing formal or cryptographic framework able to produce proofs concerning the traffic analysis resistance of mixnets.

3.2.2. Description of cMix

One of the most recent mixnet to date is the cMix protocol [Cha+16]. It is a cascade synchronous batched mixnet. Although it is not said explicitly, the firing condition of the

mix seems to be based on a threshold on the number of incoming messages (a so called *threshold mix*). The protocol requires a setup phase, in which mix nodes must interact altogether. Indeed, the protocol makes use of the Elgamal scheme as a *multi-party homomorphic scheme*, using what Section 2.5 of this thesis calls the key homomorphic property of the scheme. Each mix node M_j independently generates a key pair (pk_j, sk_j). Then, they all collaborate to produce a *system public key* $mpk = g^{\sum_j sk_j} = \prod_j pk_j$. Later in the system, there will be ciphertexts encrypted under mpk, which necessitate all the keys sk_j (and thus the collaboration of all mix nodes) to be decrypted.

The system model of cMix also includes a *network handler*, that receives the users' messages and manages them. Prior to any communication, each user X_i must perform a DHKA with each mix node M_j in the cascade, and derive multiple secret keys $k_{i,j}$. Consequently, when X_i wants to send a message m_i, it submits $m_i \cdot \prod_j k_{i,j}^{-1}$ to the network handler, *i.e.* X_i *blinds* its message with the product of all the keys it shares with the mix nodes. When the network handler has enough messages to fill a batch, it starts a *round* and notifies the mix nodes.

A round is divided in a *pre-computation phase* and a *real-time phase*. The former can be performed *asynchronously* in prevision of future rounds, and involves expensive public key operations. The latter realises the actual forwarding of batches of messages, and simply consists in (component-wise) multiplications of vectors. Indeed, cMix represents batch of messages as a vector in which each slot corresponds to one message sent by one particular user. The *shuffling* of messages is performed as a permutation on the elements of this vector: abstractly, messages of users are input in a vector $M = (m_1, \ldots, m_n)$, and the output of the network is $\pi(M)$, the vector M on which the permutation π was applied. Permutation π is the result of the composition of the individual permutation π_j of each mix node M_j.

The pre-computation phase for *one* message batch unfolds as follows, assuming the batch has n slots. First, each mix node M_j samples two random numbers $r_{i,j}$ and $s_{i,j}$ for each slot. Each M_j then communicates to the network handler

$$\mathsf{Enc}(R_j^{-1}) \coloneqq (\mathsf{Enc}(mpk, r_{1,j}^{-1}), \ldots, \mathsf{Enc}(mpk, r_{n,j}^{-1}))$$

We abuse the Enc notation: for short, $\mathsf{Enc}(R_j^{-1})$ denotes the component-wise encryption of vector $R_j^{-1} = (r_{1,j}, \ldots, r_{n,j})$. Then, for $\odot$ the component wise product of vectors, the network handler computes

$$\mathsf{Enc}(R^{-1}) \coloneqq \mathsf{Enc}(R_1^{-1}) \odot \cdots \odot \mathsf{Enc}(R_m^{-1}) = (\mathsf{Enc}(mpk, r_{1,1}^{-1} \cdot r_{1,2}^{-1} \cdot \cdots \cdot r_{1,m}^{-1}), \ldots)$$

This value is sent back to the first mix node M_1. In a second step (still in the pre-computation phase), each mix node in order permutes the vector $\mathsf{Enc}(R^{-1})$ with a randomly sampled secret permutation π_j, and multiplies the result by $\mathsf{Enc}(S_j^{-1}) = (\mathsf{Enc}(mpk, s_{1,j}^{-1}), \ldots, \mathsf{Enc}(mpk, s_{n,j}^{-1}))$. The result is the vector $\mathsf{Enc}((\pi(R) \odot S)^{-1})$ with each component independently encrypted under mpk. This result is sent to the network handler. Lastly, each mix node *commits* (using a commitment scheme) to its decryption shares of vector $\mathsf{Enc}((\pi(R) \odot S)^{-1})$. This last step is used to prevent *active* attacks, and check that the permutations in the pre-computation and real-time phase match.

Now, the real-time phase starts from a (full) vector of *blinded* user messages,

$$M' \coloneqq M \odot K^{-1} = (m_1, \ldots, m_n) \odot (\prod_j k_{1,j}^{-1}, \ldots, \prod_j k_{n,j}^{-1})$$

First, each mix node sends $r_{i,j} \cdot k_{i,j}$ to the network handler (the keys are masked by the randomness of the $r_{i,j}$ values). With these values, the latter transforms M' into $M'' \coloneqq M \odot R$, *i.e.* the messages that were blinded by keys $k_{i,j}$ in M' are now blinded by $r_{i,j}$ values in M''. Then, the actual shuffling of messages takes place on the vector M'': all mix nodes, beginning by M_1, permute M'' with π_j (the same as in the pre-computation phase), and multiply in the vector $S_j = (s_{1,j}, \ldots, s_{n,j})$. The last mix node sends the result, $\pi(M \odot R) \odot S$ to the network handler. At this point, each mix node sends to the network handler its decryption share of vector $\mathsf{Enc}((\pi(R) \odot S)^{-1})$ obtained in the pre-computation phase. The crucial point is that, now, the network handler can compute

$$\pi(M) = \big(\pi(M \odot R) \odot S\big) \ \odot \ \big(\pi(R) \odot S\big)^{-1}$$

The network handler can then publish the shuffled vector of messages *e.g.* if a broadcast channel application is envisioned. Alternatively, messages can include a receiver address, and the network handler delivers each message to its respective receiver. Finally, cMix allows answers from receivers, realising what Chaum initially called *anonymous return channels* [Cha81]. The processing of answer messages m'_i work similarly to the forward path, where each mix node uses π_j^{-1} the inverse of the permutation used on the forward path.The cMix protocol also supports application where the same permutation is used twice or more. This enables more complex communication patterns.

The authors claim to achieve relationship anonymity[2] against an *active* adversary that observes the whole network, and that corrupts all but *one* mix node and *two* users. This is a strong security level, that authors prove with formal security arguments. However, the analysis only takes *one round* into account. It would be interesting to study the consequences of re-using the same permutation over several rounds (and their return phase), since repeated use of the same random permutation may leak information about it. Concerning usability, note that a lot of constraints are put on the users: they must wait for the input batch to fill before their message is processed, and for the batch of answer messages to fill up for the return of receivers' answers as well. And there is no upper bound on the time it can take for a batch to fill up. Also, if it is not always the same user that uses the same slot in the batch, a mechanism must be put in place for mix nodes to know which key $k_{i,j}$ to use in the real-time phase.

3.2.3. Concluding on High Latency Networks

Mixnets provide more robust anonymity than low latency protocols, but are less easy to use. These protocols can be considered as suited for those ready to give up some usability and efficiency to gain in privacy. In practice, Mixnets were used before the advent of Tor,

[2]The authors actually claim *sender anonymity*, but it does not correspond to the same notion as in this work.

mainly by so-called *cypherpunk* and early privacy-aware individuals. Today, it is still used for email exchanges, in minor proportions compared to Tor.

In relation with the informant-journalist scenario, mixnets are still not enough, though. SA and RA can be broken by internal or external adversaries. In particular, observable network edges allow the global observer to detect any sending and receiving activities. Furthermore, mixnets (especially *synchronous* ones) have an inherent flaw, that leads to high probability of complete anonymity breach (*i.e.* to re-linking senders and receivers). The idea is based on observing which senders and receivers participate in given *rounds*. By intersecting the sender sets and receiver sets over many rounds, communication relationships may be found. The attack is described in more details in Section 3.4.3. Note that the authors of cMix attest that their protocol may be vulnerable to this attack, but it is left out of the formal security analysis.

3.3. Homogeneous Networks

A solution to circumvent the fundamental limits on anonymity provided by mixnets is to deviate from the client-server architecture in favor of a homogeneous network organisation. That is, a network where all nodes are client and server at the same time. This allows them to conceal their traffic and their actions as senders/receivers in the traffic they relay for other nodes, and prevent the edges of the network from being observable. In the initial presentation of mixnets by Chaum [Cha81], each user was actually assumed to run its own mix node. This idea was questioned, for the burden it puts on low power users, and because at the time, the anonymity provided with the client-server architecture was assumed roughly the same as in the homogeneous architecture.

3.3.1. Properties of Homogeneous Networks

In this survey, homogeneous networks are defined only by the fact that they use a homogeneous architecture. A protocol in this category can otherwise be a low or high latency one.

The base idea, and advantage of homogeneous networks is well formulated by Benett and Grothoff [BG03], using an analogy with a simple web proxy. In this setting, it is the users of the proxy who supposedly obtain anonymity, but it can be seen in another manner. In the words of the authors, "the only entity which can then proceed with reasonable anonymity by using the proxy service is the actual operator of the proxy service (if they use the service from within)" [BG03].

Another argument in favor of homogeneous network is its natural capacity of scaling. Having each node act as sender and relay at the same time charges users with more work, but then instead of a handful of server handling the load of many peers, many peers serve many peers. Also, homogeneous networks are naturally *peer-to-peer* and fully distributed, which means that they resist better to legal attacks (since there is no central authority to blame or take down), make it harder for an adversary to corrupt large portions of the network, and do not necessitate trust in some directory servers.

There seem to be only a handful of existing homogeneous networks. To the best of our knowledge, only three are relevant and related to direct messaging. The first protocol to actually use the homogeneous architecture, with the *explicit* idea of hiding senders among relays is Crowds [RR98]. Subsequent protocols, Tarzan [FM02] and MorphMix [DDM03], ensure similar properties, but in a more robust manner. Still, Crowds, Tarzan and MorphMix are designed to communicate with receivers outside the anonymous network. The exit edge of the network is thus observable, and the last relay node breaks RA. Crowds does not prevent traffic analysis at all and is a low latency network. Although Tarzan and MorphMix do not completely reject message re-ordering techniques, it is not explicitly included in their designs. In definitive, all three protocols are thus considered in this thesis as low-latency ones. However, designing a homogeneous high latency protocol is not impossible *per se*.

3.3.2. Description of Tarzan

Abstractly, Tarzan [FM02] is close to Tor: it works as an Internet overlay, senders chose their sequence of relays and build circuits in a telescopic manner (contrarily to Crowds and MorphMix, who let each relay decide of the next hop), and circuit identifiers are used between each pair of nodes on the tunnel. Also, an onion structure carries the messages and allows it to change of appearance at each hop.

Tarzan explicitly aims at ensuring SA, by realising what the authors call relay homogeneity, *i.e.* have sender act as relay as well, to conceal their own end-sent messages in the traffic they relay. The authors note, however, that this property is not sufficient to conceal sending activity. Indeed, by counting a sender's incoming and outgoing packets a simple local adversary can see that a node received k messages during a given time frame, and emitted $k+1$ messages, meaning that one of the $k+1$ messages were end-sent by that node. This is a breach of the SA property as defined in Section 1.4.1: even though it remains to know *which* one of these $k+1$ messages is originated by the node, this reveals that the node is sending *something*.

To counter this, at the core of Tarzan is the system of *mimics*: each node only has a few other nodes (*e.g.* 6) with whom it is authorised to directly exchange *link* messages with. The mimic relationship is symmetric: if X is Y's mimic, then Y is X's mimic as well. In this sense, a node's mimics can be considered as its neighbors in an incomplete topology graph. The idea in Tarzan, is that each node keeps a steady message flow with all its mimics, thus *protecting* these links against traffic analysis (a method also known as *link padding* [DDM03]). Namely, dummy link messages are used when a node does not have any real messages to send (*i.e.* to end-send or relay) at that time; and a node refrains from sending (end-sending or relaying) too many real messages over a short period of time. These two techniques respectively prevent a node from appearing as a clear sink and source of traffic. For this to work, link encryption ensures that dummy and real messages are indistinguishable, so that an external observer can not tell whether a message outgoing from a node is end-sent, relayed, or a dummy. To ensure that messages always follow a path only made of protected links, a node must choose the relays of its circuits according to mimic relationships. That is, an end-sender must chose its first

relay among its mimics only, its second relay among the first relay's own mimics, and so on. The last relay then sends the message to the designated receiver, such as a web server out of the Tarzan network.

The protection of mimic links is designed carefully in Tarzan, so as to minimise the observable changes in traffic pattern between two mimics, whether real messages need to be exchanged or not, at high or low rates. Performances are also taken into account. The most secure solution would be for mimics to exchange messages at constant periodic interval, in the same manner as a *heartbeat*. Although this unconditionally hides the traffic rate of a node at all times, it poses very severe constraints on the nodes and the network as a whole. Also, Tarzan aims at reasonable latency, allowing web surfing. Therefore, Tarzan constraints the outgoing traffic of a node to be loosely equal to its incoming traffic, and to be distributed among its mimics. More formally, for a node X with k mimic M_i, define $T_I(M_i)$ as the incoming rate of traffic from M_i to X. Similarly, define $T_O(M_i)$ as the outgoing rate, from X to M_i. Let $T_I = \{T_I(M_i) \mid \forall M_i\}$ be the multiset of incoming traffic rates, and f be the 33^{rd} percentile function. Then, the relations between incoming and outgoing traffic of node X are determined by the two following equations:

$$\forall\ M_i,\ f(T_I) \leq T_O(M_i) \leq max(T_I) + \epsilon \tag{3.1}$$

$$\forall\ M_i, \forall \text{ circuit } c,\ f(T_I) \geq T_O(M_i \text{ on circuit } c) \tag{3.2}$$

The first equation is a protection against external adversaries (who can not distinguish real from dummy messages), and the second against internal ones (who are able to make the distinction). The lower bound in eq. (3.1) states that a node must maintain a minimal amount of traffic (real or dummy) towards each of its mimics. The upper bound is what limits nodes in their sending rate: nodes can not suddenly decide to augment their traffic rate arbitrarily. It is allowed to have high outgoing traffic rates only if one of its mimics provides a high incoming rate. The second equation limits the sending of real messages (not taking dummy messages into account) in specific circuits. This prevents the corrupted next relay of the circuit from detecting X as sender, while allowing sending at reasonable rate even when a 3^{rd} of X's mimics are slow. More abstractly, these equations mean that mimics provide cover traffic to each other; that a node can *redistribute* high incoming traffic rates from one mimic to any other mimic; and that a node with k mimics may have a total outgoing rate k times greater than its incoming rate, by sending out to each k mimics at rate $max(T_I)$. This last point implies that an adversary corrupting two or more of a node's mimics can detect this traffic amplification, and possibly infer when a node is end-sending or not.

Finally, Tarzan provides a way to learn about all the other nodes in the network though a simple *gossip* protocol. Mimics are then chosen deterministically (and in a verifiable manner, to detect malicious nodes) from the set of all nodes. Note also that Tarzan, like the work, in this thesis, works over the Internet but effectively assumes an incomplete topology graph. This is motivated by the need to *protect* links used to exchange messages, and the cost of such a protection: the number of links to protect in a n node complete topology graph is n^2, while it is only $6n$ with the mimics mechanism (if each node has 6 mimics).

The protocol aims at achieving SA in the same version as in the present work, and at thwarting traffic analysis. The latter goal is hard to assess, but seems only partially achieved. In particular, Tarzan does not use traditional mixnet-based message re-ordering techniques, and the onion structure varies in size depending on the position of the relay in the tunnel (letting each relay estimate their position in the tunnel, contrarily to Tor). Indeed, material for per-hop integrity checking is included at each layer of the onions. The author study the anonymity of senders with a methodology inspired from Crowds. That is, they estimate the confidence (or probability) p of the adversary in finding the initiator of a circuit, based in particular on the information that she has from the size of the onions. To have *probable innocence*, meaning $p < 1/2$, the authors conclude there must be less than 40% of corrupted nodes [FM02, Fig. 7]. However, this analysis is informal (from a cryptographic point of view), and implies strong assumptions on the adversary (see Sections 5.1.2.a and 6.4 for further discussion on this point).

Lastly, note that Tarzan is not subject to the same attack as mixnets, since the entry edge of the network is not observable. If the protocol may realise SA in a robust manner because traffic analysis is still possible, it at least makes a step towards it: instead of the first relay *always* breaking SA, as in Tor and cMix, Tarzan introduces some uncertainty in the detection of senders.

3.3.3. Concluding on Homogeneous Networks

Ensuring *relay homogeneity* is what realises the homogeneous architecture. But its implementation seems to put a lot of load on users, and to consume bandwidth, because of the need of dummy messages and controlled traffic rates. A homogeneous *high latency* protocol is likely to be even less efficient than a mixnet, but can still be easier of use. In particular if the synchronous cascade approach of cMix is rejected in favor of a more flexible mixnet design. Although this gain in flexibility translates in a loss of security and privacy (meaning that traffic analysis may become easier), we argue that relay homogeneity can fill this gap, by making traffic analysis much harder, especially for external adversaries.

Taking a step back, the particularity of homogeneous networks lies in the active participation of all users in order to achieve the anonymity that it provides as a whole. On the other hand, non-homogeneous networks are meant to provide anonymity *as a service*, and do not require much from the user (apart from patience with regards to the added latency). A homogeneous network is thus a sort of solidarity network, and is in this sense designed for a public ready to spend resources for the community as a whole. It is suited for our journalist-informant scenario, and this is the network architecture used in this thesis.

3.4. Review of Known Attacks

During the description of low, high, and homogeneous networks, several attacks were mentioned or referenced. This section reviews existing attacks in more details. The aim is to obtain a clear view of the threats to anonymity, and of possible defenses. Because

a passive adversary model is considered, only passive attacks are listed here. The list here is obviously not exhaustive, be it only because new attacks are discovered on a daily basis.

Passive attacks against anonymity properties mostly rely on observations, collection of information, and analysis of this information. The goal often comes down to linking a (link or payload) message with a particular end-sender or end-receiver. In this sense, most attacks are related to traffic analysis. Analysis can be based on individual messages, or on full sessions (that is, on the *flow* of messages in communication sessions or in connection-based protocols such as TCP). Roughly, the sources of information to the adversary are: the appearance of messages, the timing between messages, and their number. These information sources may not suffice by themselves. Typically, a more advanced attack consists in observing *many*[3] messages and/or nodes, and by *intersecting* the information gathered from these observations. This allows to carry out statistical analyses, and to check assumptions, *e.g.* on communication relations. The goal being, eventually, to break SA, RA, or even completely re-link senders and receivers. Although this generic description is valid for many attacks, there are of course some network-level or cryptographic attacks that use different methodologies.

This section begins with a description of what can be learned by simply looking at the appearance and *bit pattern* of messages. The rest of the section describes more advanced attacks, taking place at the system or network level. In particular, we present the inherent flaw of non-homogeneous mixnets, which allows to re-link senders and receivers. The last parts focuses on advanced flow-based timing analysis and traffic fingerprinting attacks. In conclusion of this section, Table 3.1 summarises which of the Tor, cMix and Tarzan protocols are vulnerable to which attacks, and reviews the privacy goals of this thesis in light of what was learned.

3.4.1. Attacks based on Appearance of Messages

The most basic source of information that the adversary has is the *appearance* of messages, meaning their *bit pattern*, the information they contain, and all cryptographic material that may be associated to them. Either as a network observer or a corrupted node, a message's appearance can in particular allow its tracing, and ultimately lead to finding the end points of communications.

Firstly, the most trivial breach of privacy is when the message contains information *in the clear* about the route or communicants. The worst case being when the identity of the end-sender or end-receiver appears, as in the Crowds protocol, where the receivers' identity is visible to all relay nodes on the path. Another, less trivial case, is when messages carry information on the route length, on the location of the relays that process it on the route, or on the number of passed or remaining hops. Access to such information can bring quite a lot of information to the adversary. In particular, Berthold *et al.* [BPS01] show how this can greatly degrade privacy in asynchronous free-route networks. Most anonymous protocols are of course designed to carry in clear only the minimum necessary for the

[3] *Polynomially many*, to be more accurate.

protocol to work. External observers in particular, often do not get any information.

Then, another trivial issue arises when a message does not change of appearance as it is relayed. Whether the message is encrypted or not, if its appearance is the same from one end of the route to the other, its tracing is trivial. In Crowds again, relay nodes do indeed all see the exact same data (encryption is only used *between* relays). In most anonymous protocols, the appearance of messages is thus changed, even from the point of view of the relays. This is performed using, *e.g.* onion encryption or URE.

Still, those changes of appearance may be predictable in some way. In particular, in mixnets based on onion routing, if the exact same message with the same onion structure is sent on the same path on two occasions (*i.e.* in two different batches), it will follow the exact same sequence of transformations, and thus be easily spotted and traced. A typical solutions is for relay nodes to discard any already seen messages. Another solution is to use URE: because of its probabilistic nature, the same incoming message (*i.e.* the same ciphertext) will yield a different re-encrypted outgoing message each time.

However, another threat arises with URE, that can be described as recognising a message based on its encrypted contents. In a protocol where messages are simply encrypted under the end-receiver's public key, and re-encrypted at each hop, a corrupted end-receiver (coupled with a network observer) can perfectly trace all messages meant to itself. For that, it needs only to decrypt the ciphertexts in the messages, and access its contents, that stays constant from the end-sender to the end-receiver. A simple counter-measure is to add link encryption, making the attack only possible to corrupted relays working with Bob. Alternatively, messages can be encrypted under a key other than the end-receiver's, or under a product of public key. For instance, in the already mentioned protocol by Huang *et al.* [HLF12], a cascade mixnet that uses the URE-Elgamal scheme, a message m for end-receiver R going through mix nodes M_j, $j \in [1, k]$, is encrypted by the end-sender as $c = \mathsf{Enc}(pk_R \cdot \prod_j pk_j, m)$. Each mix node on the path then *partially decrypts* and re-encrypts the ciphertext, so that its comes out of the network encrypted solely under pk_R).

Independently from the contents and appearance, messages can be distinguished or traced based on their sizes. More exactly, there are two dimensions: different end-senders could provide different sized messages, or messages' size could change as they evolve in the network. The first possibility is especially devastating in a batched mixnet: in the worst case, if messages constituting a batch have different sizes, the whole purposes of shuffling is defeated, and each message in the batch can be traced trivially. The second possibility can be illustrated with the onion structure of the Tarzan protocol, which diminishes in size at each hop of the circuit. This leaks information to relay nodes on the length of the circuit, and/or on its own position in the circuit (thus coming back to the problem of leaked hop count). Therefore, anonymous protocols often require that all messages have the same size, or at least split long messages into fixed-size units (*e.g.* 512 bytes in Tor). Then, to ensure constant size of the messages as they travel, protocols either use a fixed-size data structure (*e.g.* Tor encrypts data *in-place*); or, add padding to always match the maximum size of the message on the route. For instance, Mixminion uses a onion structure that grows in size, but adds padding that even relay nodes can not distinguish from actual (encrypted) data.

Summing up, an anonymous protocol should at least include the basic elements of design that: conceal most of the information carried by messages, change appearance of messages at each hop (preferably in a probabilistic way), and ensure uniform size of all messages. These should hold both for network observers and (corrupted) relay nodes.

3.4.2. Network Discovery and Relay Selection Attacks

In anonymous networks, homogeneous or not, users must be able to learn about other nodes in the network. In many protocols, they then choose the relays they want to use based on their view of the network. There are potential pitfalls in this process: users may choose corrupted relays, and users may have a *biased* view of the network and the nodes therein.

Choosing one or a few corrupted relays does not automatically mean a breach of privacy, since anonymous networks take into account and include defenses against this scenario. However, a protocol such as MorphMix, who lets each relay select the next hop, is subject to *route capture*: if at some point, a corrupted relay is selected, it is likely that the message will then be routed only through other corrupted relays from the same collusion. Against this, MorphMix includes a collusion detection mechanism, based on the idea that corrupted nodes within a collusion appear more often together in anonymous tunnel than honest nodes. Another threat arises when the adversary is able to place itself in a position where its relays are more likely to be selected by users. For instance, Tor's standard relay selection biases the users' choices towards relays with high bandwidth. While it is good for efficiency and for the functioning of the network as a whole, it is likely that the adversary has the required resources to run powerful onion routers, and capture users' circuits. Recent works show improved anonymity guarantees for more *uniform* choices of relays [BMS16].

A second type of threat arises when the user has partial or out-dated information on the network [DS08]. More generally, when different users have different views of the network, the adversary can exploit these discrepancies to partition and distinguish users. In a protocol using directory server [DDM03; DMS04], it may be the case that a server S is listed only on directory server D but not on others. The adversary can thus attribute messages forwarded by S only to users that have queried D, degrading privacy [DDM03, Section 7]. The same attack applies if users download only a fraction of the directories. Another example, applied to completely distributed networks with no directory servers, can be found in an early design of Tarzan, where nodes learned only a random fraction of nodes in the network. Consequently, the circuit built by individual users could be attributed to them based on the uniqueness of the information they had learned [DC06; DS08].

To remedy to the above attacks, Tarzan opted for a full network discovery, at the expense of decreased efficiency and scalability. For protocols based on directory servers, all directories should at least have the same information at all times, and ideally, users should always download the whole directories (to hide which nodes they are interested into). This is the design of directory servers in Mixminion [DDM03] as well as in Tor [nic12]. Additionally, because directory servers are a target of choice for the adversary, they

should be well secured, and monitored (*e.g* using a reputation mechanism).

3.4.3. Limits of the Mixnet Model

Section 3.2 made a reference to a fundamental flaw of mixnets, especially synchronous ones. That is, mixnets are subject to an attack that allows a complete re-linking of senders and receivers.

The first version of this attack, hereby called the long term intersection (LTI) attack, is due to Kesdogan *et al.* [Kes+06]. The authors demonstrate the maximal theoretical anonymity mixnets can provide using information theory. Their main result states that an adversary observing the senders and receivers at the edges of the network, can *theoretically* perfectly re-link all senders and receivers across all communications, even if the mixnet is considered a perfectly secure black box. This result is mainly applicable to batched synchronous mixnets, where the mixnet functions in separate *rounds*. This mixnet variant was believed the most robust, since attacks had previously been found against the other types of mixnets, asynchronous ones [DSS04] and continuous ones [Dan04]. Now, the situation is not as clear, since each mixnet variant has its own weaknesses.

This theoretical result rests on the fact that not every sender and receivers participate in each round: sender and receiver anonymity sets differ from round to round, and by intersecting them, the actual mapping between senders and their respective receivers (the receivers it has communicated at least once in the system) can be completely found. It is possible to prevent this, by having all senders send a message in *every single* batch round, and by broadcasting the output of the mix to all receivers, making sender and receivers sets always maximal. But this solution is totally impractical: for payload messages of 4KB and $n = b = 10,000$ users, each receiver is given $400,000$MB [Kes+06, Section 2.2].

The result of Kesdogan *et al.* is only theoretical because it assumes a computationally unbounded adversary having access to all the observations it could ever need. But several practical attacks were proposed, broadly following the idea of intersecting observations. A good history and comparison of practical attacks can be found in [PTO14]. In a nutshell, the first attack [KAP02] was computationally expensive (requiring to solve a SAT instance), assumed very simple sender behavior, and simple batching strategies. Following works made weaker assumption, always increasing the accuracy of the attack, and considering more realistic sender behavior, and more robust batching strategies. Latest developments study the effect of *pool-based* batching strategies (where a message may stay in the mix for more than one round) [PTO14] and dummy messages [OTP14] using a *least square errors* (LSE) approach, which is the most efficient one to date. This approach consists in the adversary making assumptions on the sender-receiver relationships, under the form of probability vectors. After observations, is retained the probability vector that minimizes the difference, in the sense of LSE, between (i) the observed output messages on one hand, and (ii) the combination of the observed input messages with the assumption of the adversary on the other hand. The attack does not need to enumerate all hypotheses, rather to start from one hypothesis, and from there, progress towards a local optimum.

Generally, results show ([KAP02, Claim 1], [Dan03b, Equation (4)], [Kes+06, Sec-

tion 5.3], [PTO14, Section VI.C]) that the success of attacks for a fixed sender behavior and batching strategy, depends on: the number r of rounds observed, the total number of users n, the batch size b, and the number m of receivers each sender communicates with. As r augments, the attack's results get more accurate (and are completely accurate when $r \to \infty$). Also, the attack gets easier when n augments, b decreases, or m decreases. As an example, the practical attack studied by Kesdogan *et al.* [Kes+06] needs no more than 600 observations for its attack to succeed, for $n = 200,000$ users, batch size $b = 100$, and $m = 40$. Also, the sender behavior plays a role: discrepancies among senders behaviors helps the attack. Notably, a sender that communicates much more than the others is easier to target [OTP14, Fig. 2]. But the LSE approach works for any behavior that can be probabilistically modeled. The batching strategy is also a factor. It seems that pool-based mixes resist better, and that the longer a message may stay in the pool of a mix, the harder it is to link senders and receivers [PTO14, SectionVI.D] (at the cost of longer latency in delivery of course).

Because all the above attacks assume that it is possible to observe senders and receivers post or get their messages from the network, they can not be carried out when SA or RA is ensured. This is the idea used in Tarzan and homogeneous protocols more generally, and put forward in this thesis.

3.4.4. Detecting End-Sending and End-Receiving Activities

In a non-homogeneous network, observing the network edges allows to detect end-sending and end-receiving activities trivially. However, even homogeneity is not sufficient in itself, as already noted in the analysis of Tarzan (Section 3.3.2). If the adversary can observe all the links of a given node, she can see the numbers n_i and n_o of incoming and outgoing messages of a node during a given time frame. And if $n_i > n_o$ or $n_i < n_o$, the adversary concludes that the node end-sent or end-received at least one message. This is a breach of the SA and RA properties as defined in this thesis.

Typically, a protocol prevents these breaches of privacy not only by endorsing a homogeneous architecture, but also by introducing dummy messages. As in Tarzan, the idea is to blind the n_i and n_o values and making them an unreliable source of information for the adversary. Of course, for that, dummy messages must be indistinguishable from real messages. However, in Tarzan, dummy messages are (meant to be) differentiable from real messages by relay nodes, and thus have limited effect on internal adversaries. In the worst case, when a node is surrounded by corrupted nodes, the types of dummy *link messages* are completely ineffective [Boh+04]. This last fact is one of the reasons to be of Assumption 3, that every node has at least one honest neighbor. Alternatively, some works use dummy *end-to-end messages*, indistinguishable from real messages even for relay nodes (only the end-receiver makes the difference). The cost of end-to-end dummy is greater than simple link ones, and their effectiveness is questionable, in particular in view of the advanced timing and traffic fingerprinting attacks described next.

3.4.5. Timing Analysis

Timing analysis, in the general sense, may be the most complex and less understood class of attacks in anonymous networks. Yet, it is recognised as an important threat, allowing ultimately to re-link senders and receivers [Dan04; MD05]. It is most effective against low latency protocols, but can also be applied to continuous mixnets. It efficiency against batched mixnets is limited, although not null [Zhu+04].

Timing analysis is based on the assumption that a message entering a node at time t will leave it at a predictable time $t + \epsilon$. This principle can also be ported to a sequence of nodes, or even by looking at messages entering and leaving the network (if it is not an homogeneous one). More generally, timing analysis is conducted on communication sessions and flows of several messages, by looking at *inter-message intervals*: in a low latency network with no message re-ordering of any kind, if two messages leave some upstream node X with δ time difference, they will with high probability arrive at downstream node Y with time difference $\delta \pm \epsilon$ for a small ϵ. To carry out a timing analysis, the adversary must be able to observe messages. It is easier to do so is she has corrupted nodes in the network, but in some cases a simple network observer is enough, in particular, when no dummy messages are used or when network edges are observable.

Practical attacks do not directly use the inter-message intervals, but divide the time into fixed-size windows and counts the number of message observed during each window [Lev+04; SW06]. Shmatikov and Wang [SW06] study the possibility of correlating *sparse* flows (that is, link a given flow entering the network, with an exiting flow) in low latency networks. Sparse flows have the particularity of alternating noticeable bursts of traffic and low traffic rate (*e.g.* TCP of HTTP connections). Results show that nearly all flows can be correctly re-linked by the adversary. In addition, Levine *et al.* [Lev+04] make the experience with a steady traffic rate, and show that, although the traffic does not present significant *bursts* as in the previous attack, it is still possible to successfully correlate flows. Of course, these attacks are more accurate as the number observed of messages in flows increases. Danezis [Dan04] extends the attack to continuous mixnets, where each message is independently delayed a random amount of time. In this case, the observed inter-message interval is not the same at the entry and exit of the network, since random delays are introduced. However, the attack still works, because δ' is a predictable function of δ, that can be estimated from the probability distribution of the random delays added and the number of relays on the paths. Again, it is possible to correlate a large portion of entry and exiting flows.

The presented attacks work because flows conserve their *timing signature* as they traverse the network. And the more sparse the flow is (with *silent* phases interleaved with sudden bursts of traffic), the more unique its signature is, and the easier timing analysis gets. Therefore, defenses against timing attacks mainly consists in disturbing that *timing signature* of flows. One way to do so is to use batched mixnets, perturbing timing signatures in less predictable ways than continuous ones. For works aiming at low latency, other ideas include *defensive dropping* [Lev+04], where some dummy messages are relayed on several hops and dropped randomly at some point, or *adaptive padding* [SW06], where messages are inserted opportunistically in between bursts in the

flow. Note that homogeneous networks are not subject to end-to-end traffic analysis since network edges are not observable, but leaves the possibility of timing analysis by corrupted nodes along the same (portions of) routes.

3.4.6. Traffic Fingerprinting and Application Layer Information Leak

The last class of traffic analysis reviewed in this chapter is traffic fingerprinting. It is an advanced attack, akin to timing analysis, but based on multiple criteria to detect the *signature* of a flow. It additionally uses metrics such as the number of messages, their order, their size, or any other elements that characterises a flow, in addition to timing information. Such a collection of characteristics is hereby called the *fingerprint* of a flow. In particular, there may be a lot of information that leaks from the application layer. Indeed, even if the contents of application message is encrypted and/or sanitized of sensible information about the end-sender for instance, the fingerprint of a SSH flow often greatly differs from a HTTP flow; and the flow for a specific web page will be different than that of another page (depending, notably, on the additional resources such as images that needs to be downloaded). This type of attack is a real threat, in particular to low latency networks and continuous mixnets. Several attacks on the Tor network have been shown effective in practice [Jua+15; Gha16]

There are many practical attacks in the literature, summed up by Ghaleb [Gha16, Section 2.1], each with different strategies. Most often, attacks are based on machine learning, where a classifier must be trained on a set of known flows. Then, it is given a flow from some anonymous protocol, and makes a guess. In Tor, this idea is most often applied to distinguishing which web pages are visited by users, an attack called *website fingerprinting* [Jua+15], that exploits the differences in web pages sizes and resources. The relations between HTTP flows can also be leveraged by the adversary: she can use the fact that when some web page, say, `www.example.com/index.html`, is visited, the page `www.example.com/about.html` will be visited shortly after.

It is interesting in this thesis to look at how traffic fingerprinting performs against high latency networks, such as batched mixnets. Zhu *et al.* [Zhu+04] are among the few works that study traffic fingerprinting of batched mix nets with several batching strategies (including the pool-based ones). The authors show that, it is possible to distinguish a FTP flow from a flow of dummy messages. With more than $20,000$ messages in the flow, the probability of distinguishing approaches 1. However, these experiments considers an extremely simple scenario, with solely one mix node, and only two flows going through it. Also, the parameters of the batching mechanism is extremely low: batches are of roughly $b = 10$ messages, and the firing condition consists in flushing the mix node every $t = 0.01$ seconds. This explains why the studied network is able to handle a FTP flow, normally incompatible with high latency networks. In comparison, one of Mixminion's practical implementations uses a time interval of the order of the minute *e.g.* $t = 15min$ [Mat11]. In definitive, this study mainly shows that low batching parameters leads to insecure mixnets, and inspires care: batched mixnets may still leak a small amount of information on flows.

Several counter-measures were proposed [Gha16, Section 3.3], in the same idea of

defenses against timing analysis: disturb the flows' fingerprints. Most works aim at doing so while introducing the least latency as possible in the delivery of messages. The *BuFLO* mechanism [Gha16, Section 3.3] proposes to make all flows look the same (at the cost of some latency and bandwidth), while *traffic morphing* tries to prevent website fingerprinting by altering flows to imitate another web page. Also, Juarez *et al.* [Jua+15] re-use the adaptive padding technique from Shmatikov and Wang [SW06]. Serjantov and Murdoch [SM05] propose to tackle the base assumption made by almost all traffic analysis attacks (including timing ones) that all messages of a flow go through the same path. By splitting messages from a same flow, they show that anonymity can be improved. However, we note that splitting flows is hard to achieve for connection-based protocols, such as TCP, which are sensible to messages order and latency. But it is possible for direct messaging, where a *flow* is simply a large message split into chunks.

3.4.7. Concluding Remarks on Attacks

All the above attacks were presented in a generic manner. However, it is important to remember that most of them apply to specific types of networks, or are based on explicit assumptions. For instance, the LTI attack works only if network edges are observable. Also, attacks can generally be made even more effective if the adversary has some *a priori* knowledge, or already suspects that some entities are communicating.

Table 3.1 intends to summarise which of the presented protocols (Tor, cMix and Tarzan) are vulnerable to which attacks. A "✔" means the protocol resists the attack, "✗" that it does not. The "?" for the LTI attack w.r.t. Tor denotes an uncertainty. Indeed, this attack mainly applies to batched mixnets, but may also work on (non-homogeneous) low latency networks, by considering *time windows* instead of batches. More generally, this table is informal, and a thorough study of each attack applied to each of the three protocols would be needed to validate it.

		Tor	cMix	Tarzan
Appearance	Contents	✔	✔	✔
	Hop Count	✔	✔	✗
	Enc. Contents	✔	✔	✔
	Chg. Appearance	✔	✔	✔
	Size	✔	✔	✗
Obsrv	Senders	✗	✗	✔
	Receivers	✗	✗	✗
Relay Selection		✗	✔	✔
LTI		?	✗	✔
Timing Analysis		✗	✔	✗
Traffic Fingerprinting		✗	✔	✗

Table 3.1. – Vulnerabilities of Presented Protocols

Taking a step back, these attacks show that preventing (advanced) traffic analysis in an

efficient manner is still an open problem. To do so while maintaining low latency seems near-impossible, and even mixnets, which abandon some efficiency to gain robustness, may not be immune to traffic analysis. In particular, the guarantees brought by dummy messages and message re-ordering mechanisms are uncertain (in particular, they can not be formally proved, as of today), while the burden they put on the network may be high [DP04; DMS04; OTP14]. Shamatikov and Wang rightfully remark that any defense is ultimately vain if flows follow different statistical distributions, by stating that "even small statistical differences between packet flows can be detected if the observation time is long enough" [SW06, Section 5.1]. In this light, the possibly only robust defense would consist in sending messages at a constant rate (and send dummy messages when no real messages need to be sent), and thus make all flow have the exact same fingerprint. Actually, this design may be the only one that is *provable* in the cryptographic sense, as resisting against any PPT adversary. But this *trivial* solution is much too costly, even for high latency networks. The idea, as in the whole field of computer security, is ultimately to find a middle ground, where the time, resources, and number of observations needs to the adversary is high enough compared to the envisioned use of the network.

In view of all these attacks, where do the SA, RA, SU, MU and TAR properties stand? Clearly, all attacks ultimately aim at breaking SA and RA. And for that, they tackle SU, MU and/or TAR. Ensuring SU prevents, in particular, the traffic fingerprinting attack based on the linking of several HTTP flows (the one for the web page, plus the ones for its linked resources). More generally, it *segregates* the knowledge the adversary acquires about each session or flow. The review of attacks also makes the differences between MU and TAR appear more clearly. MU essentially relates to the attacks based on the *bit pattern* of messages (described in Section 3.4.1), while all the others relate to TAR. This seems to imply that MU is ensured by cryptographic means and can often be formally studied and proved, while TAR can not.

3.5. Summary: Where this Thesis Stands

This survey of existing privacy-preserving protocols, and of the different attacks on anonymous network, puts in light some leads to further improve network users' privacy. It appears firstly that ensuring SA and RA, in the strong sense given in this thesis, is not usually among the primary goals of existing constructions. Yet, for our envisioned informant-journalist application in particular, it is crucial to prevent the observation of end-sending and end-receiving activities. From the example of the Tarzan protocol, it appears that realising SA and RA implies a homogeneous architecture, and the use of dummy messages and controlled traffic rates. However, Tarzan's mimics mechanism does not seem strict enough, since it still allows a node to suddenly augment its end-sending rate, making it easy to detect end-senders for the adversary. We propose an adaptation of this mechanism that abstractly aims at making each node appear as if it was only ever relaying traffic for other nodes, even in the presence of corrupted neighbors and external observers.

Secondly, in order to allow the journalist to communicate with the informant without learning her identity, we have seen Tor's *hidden services* solution, that makes use of pseudonyms and *rendez-vous* points. In this thesis, the informant's anonymity w.r.t. the journalist is ensured by mechanisms inspired from both these techniques.

Lastly, from the review of existing threats to privacy, it appears that ensuring the TAR property in a robust, formally verifiable manner is extremely challenging or even impossible. Yet, timing analysis and traffic fingerprinting are serious threats, since they are stepping stones towards breaching SA and RA. Still, there is hope. Overall, past works indicate that the most traffic analysis-resistant networks are the pool-based mixnets, that come with complex firing conditions [Mol+03], and where batches coming out of a mix node are not solely composed of messages that recently entered the mix, but of potentially any message received by the mix since the start of the network. Also, the most serious attacks, timing analysis and traffic fingerprinting, often make the assumption that both edges of the network are observable, which is not possible in a homogeneous network where SA and RA are ensured. However, the challenge here is to adapt pool-based mixing to the setting of fully distributed, homogeneous networks.

4. The Anonymous Protocol

This chapter presents our homogeneous protocol ensuring sender anonymity (SA), receiver anonymity (RA), message unlinkability (MU), session unlinkability (SU) and traffic analysis resistance (TAR) over the Internet. The security of the protocol is analysed and proved in Chapter 5, while Chapter 6 presents a prototype implementation of the protocol, and studies its practical performances and privacy guarantees.

At a high level, the protocol is an adaptation of Tarzan [FM02] to the mixnet setting, preventing the very detection of the actions of end-sending and end-receiving. It differs from usual sender-built circuits, since instead of building circuits *on-the-fly* and when needed, circuits are long-lived and built in a proactive manner. That is, the protocol comprises a *topology dissemination* phase, where the nodes learn about their extended neighborhood, and ultimately every other node. Also, the protocol consists in a shift in terms of identity management: to enable end-receivers to be anonymous even w.r.t. to end-senders, nodes are identified in the network with *relationship pseudonyms* [PK01].

The chapter is organised as follows. The first section is a didactic overview of the protocol, effectively summing up the whole chapter. The overview also presents the motivations and consequences of using relationship pseudonyms. Building on the overview, and following a similar outline, the complete protocol presentations spans over Sections 4.2 to 4.5. Section 4.2 begins by presenting the routing tables as obtained after the topology dissemination phase, and Section 4.3 details how these tables are used, and the cryptographic processing of messages by nodes. Section 4.4 is devoted to the description of the dummy messages, controlled traffic rates and message message re-ordering mechanisms, that play an important role in ensuring SA, RA, and TAR. Section 4.5 describes in details the topology dissemination, and the construction of routes and routing tables. Section 4.6 presents the final building block, oriented communications, where an informant anonymously contacts a journalist by leveraging the properties of the pseudonyms. Finally, Section 4.7 summarises some of the interesting properties of the protocol, and concludes

4.1. Overview

This section is a summary and an introduction to the full protocol description. It is organised as the whole chapter, and acts as a condensed version of it.

The protocol works over the Internet, and is based on a homogeneous architecture. It assumes a *sparse* underlying topology graph [Dan03a] (an incomplete but connected graph), where each node knows only its direct neighbors' IP address. How this underlying topology graph is constructed is out of the scope of this work.

The protocol belongs to the *high latency* category, for it aims at TAR and uses reordering techniques for that, under the form of pool-based mixing. It can be described as a *restricted-route* mixnet [Dan03a] (*i.e.* not a cascade, but not a free-route network either), and functions in an asynchronous manner. The routes are long-lived circuits built starting from end-receivers. A circuit relates to only one end-receiver, but several end-senders share the same circuit. Neither end-senders, relays, or end-receiver know which nodes are part of a given circuit: the insider's knowledge is limited to the previous and next hop. The protocol encrypts messages under a *product* of public keys (including the end-receiver's public key), uses URE to change the appearance of messages, dummy messages to prevent the observation of the network edges, and controlled traffic rates to protect against corrupted neighbors. Last but not least, nodes designate each other using pseudonyms instead of their IP addresses or real-world identities.

Relationship Pseudonyms

The protocol proposes a shift from traditional identity management, by making nodes designate each other with *relationship pseudonyms* [PK01]. Therefore, in a network of n nodes, each node has $n-1$ anonymous network identities; and nodes X and Y designate the same end-receiver node R under two completely different and unlinkable pseudonyms. This design choice, along with the goals of SA, RA, MU, SU and TAR, greatly impacts

the way the protocol is built, and implies the need to introduce mechanisms that are non-standard w.r.t. previous works.

This choice is motivated by the need for nodes to be reachable while staying anonymous even from the end-senders. In the considered informant-journalist scenario, this enables the informant to receive messages from the journalist, while remaining anonymous even to the latter. In terms of functionality, these pseudonyms can be seen as a stronger version of Tor's hidden services, where the end-sender does not know the real-world identity of the end-receiver it is communicating with.

Relationship pseudonyms provide better anonymity than traditional ones, which implies that a node is known by every actor in the network under the same unique pseudonym. It is the case with the pseudonyms used by receivers hiding behind hidden services in Tor, or those used by Bitcoin walled addresses [Nak08]. Indeed, with a traditional pseudonym, all the actions of a given node can be linked together. Ultimately, this allows profiling and easier de-anonymisation [CKK05]. On the other hand, relationship pseudonyms make coordinated attacks from several actors in the network harder to carry out. Additionally, once a traditional pseudonym is de-anonymised by the adversary $\mathcal{A}$, the latter can publicly announce the linking between the pseudonym and real-world identity. With relationship pseudonyms, it is not that simple, since the relationship pseudonym used by $\mathcal{A}$ is meaningful only to herself: announcing publicly who is hiding behind that pseudonym does not give information to other nodes. Of course, relationship pseudonyms do not prevent de-anonymisation in themselves. Rather, they *limit the consequences* of de-anonymisation. Actually, endorsing relationship pseudonym is a way to attest that some nodes will inevitably be de-anonymised at some point, despite the best efforts put in the design of the protocol, and to introduce a form of *damage control*. Indeed, even provably secure protocols are subject to de-anonymisation (in particular, by attacks out of their model). For instance, even though mass de-anonymisation of Tor user is still believed impossible, targeted ones are largely achievable.

In this work, the relationship pseudonym (or anonymous network identity) used by node X to designate end-receiver node R is denoted $PS_{X \to R}$. These pseudonyms are designed to be cryptographically secure, meaning at the very least that the real-world identity of R can not be found from $PS_{X \to R}$, and two corrupted nodes X and Y should not be able to *compare* their pseudonyms, *i.e.* $PS_{X \to R}$ should be *unlinkable* to $PS_{Y \to R}$. Section 4.5.2 provides a more detailed definition of pseudonyms. To the best of our knowledge, the use of relationship pseudonyms as defined in this work is new in secure messaging, although they have been studied as part of various privacy-enhancing identity management frameworks [CKK05; AG12].

There are consequences to this choice of identity management on the design of the protocol. Firstly, particular care is taken to avoid using any *network-wide identifier*, *i.e.* the protocol avoids to use any piece of data that would be used by the whole network to designate the same node. In particular, the public key of a node is considered as a network-wide identifier. Thus, the advertising and direct use of public keys must be avoided or circumvented. Also, circuits can not be built by end-senders that freely choose their relays, since they do not know the IP address of the end-receivers they want to address. Rather, circuits must be built starting from the end-receiver, during a phase of

topology dissemination.

Routes and Routing Tables

Every anonymous network protocol has routing tables, even if they are implicit. In the Tor protocol, for instance, nodes store mappings $\langle (OR_{prev}, cid_{prev}), (OR_{next}, cid_{next}) \rangle$, often accompanied with various cryptographic tokens. The packets abstractly consist in a circuit identifier and some (encrypted) payload data. The circuit identifiers *cid* are what allow the relay node to know where to send a message next.

In this work, routing tables additionally contain the pseudonym of the end-receiver that circuits lead to. This pseudonym is not used for relaying messages, but for end-sending. Indeed, different nodes do not designate a given end-receiver with the same pseudonym, thus it is not possible to route messages based on pseudonyms. Note that, even though we aim at concealing the IP address of end-receivers, it is unavoidable to make the IP address of the next hops appear in the clear in order for the protocol to function as an Internet overlay.

In definitive, routing tables entries and packets in this protocol, in their simplest form, respectively consist of:

$$\langle (IP_{prev}, cid_{prev}), PS_{X \to R}, (IP_{next}, cid_{next}) \rangle \text{ and } \langle cid_{next}, data \rangle$$

Sending, Relaying, and Receiving Messages

Routing tables are used by nodes in their activities as end-sender, relay, and end-receiver. But several mechanisms are put into place to ensure SA, RA, SU, MU and TAR.

Firstly, to (partially) ensure MU, and TAR, link encryption and URE are used. That is, similarly to several existing works, circuit identifiers are encrypted with a SKE key shared by neighbors, obtained by a DHKA run at network setup. This prevents external adversaries from knowing which messages belong to which circuit. While circuit identifiers change at each hop, the payload does not. The payload is thus encrypted separately from the circuit identifier, and its appearance is changed using URE. Although URE does not seem to integrate well with the traditional client-server architecture, in a homogeneous setting, it has several advantages over onions structures. It allows to simply encrypt payload a m under some (product of) public key(s), and have relay nodes re-encrypt the ciphertext at each hop, without even needing to know these public keys. In regards of the pseudonyms, it also avoids the advertisement of public keys. However, URE is used in a manner that differs from past works. Indeed, the presence of a topology dissemination phase allows nodes to learn adequate encryptions of one, thus removing the burden of always sending full URE-Elgamal ciphertexts (*i.e.* a single Elgamal ciphertext is routed through the network, instead of two). Also, to prevent the URE-specific attack presented in Section 3.4.1, payloads are actually encrypted under a *product* of public keys, and each relay node on the route divides out its own key.

More specifically, during the topology dissemination, each node X obtains a ciphertext $c_{\texttt{one}X \to R} = \mathsf{Enc}(pk_{Z_1} \cdot pk_{Z_2} \cdot \ldots \dot{p}k_R, 1)$ towards each node R, *i.e.* the value 1 encrypted under the product of all the keys of nodes along the route between X and R. With this,

X can encrypt a payload m by leveraging the homomorphic properties of the Elgamal scheme. Namely, it computes $\mathsf{PlainMult}(c_{\text{one}X\to R}, m)$. With the same ciphertext $c_{\text{one}X\to R}$, X can also re-encrypt any ciphertext that it relays for other end-senders. Note that encryptions of one, contrarily to public keys, do not constitute a network-wide identifier, since there are many possible encryptions of one for any given public key.

The above elements of design mainly participate in ensuring MU. To ensure TAR, messages are additionally re-ordered during their forwarding, using techniques from mixnets. Here, we chose to use *timed dynamic pool* from Mixmaster [Mol+03], for its robustness to traffic analysis. However, instead of maintaining one global pool of messages, a node maintains one pool for each of its neighbors. Consequently, every $t_{\mathcal{P}}$ seconds, a node checks if *all* its pools have enough messages in them, and if so, fires the mix by sending a random fraction of each pool to the corresponding neighbor.

Finally to ensure SA and RA, in addition to endorsing a homogeneous architecture, dummy messages and controlled traffic rates are used, similarly to Tarzan. Here, however, we pose more constraining rules, aimed at making a node *appear* to link-send as many messages as it link-receives, *i.e* make every node appear only as a *relay* of traffic. Basically, this translates into compensating the excess of link-received messages by sending out one dummy message *to each neighbor*. And similarly for the excess of link-received messages. Ultimately, we show that these constraints prevent the detection of sending and receiving activities *even when only one neighbor of the node is honest*.

There is one question left, however: for a given informant Alice that wants to communicate with a specific journalist Bob, how does the informant find a route towards that journalist? Indeed, the informant's routing tables contain no information linkable to the real-world identity "Bob". Said otherwise, routing tables leaves only the possibility of completely anonymous communications, where end-senders and end-receivers have no idea who they are communicating with. This may be sufficient for some applications such as anonymous file sharing or online gaming, but not for the informant-journalist scenario. Thus, the protocol also enables *oriented communication*, a way for Alice to contact Bob specifically, by leveraging the properties of pseudonyms and the way they are constructed. The proposed solution is for Alice and Bob to use an *indirection node* I in the network: Alice has the real identity of Bob, and interacts with I to compute I's pseudonym towards Bob. During the interaction, care is taken to separate the knowledge between Alice and the indirection node, so that neither Alice nor the indirection node can link Bob to its pseudonym(s).

Constructing the Routes

Abstractly, topology dissemination is quite standard: as in every self-discovering network, nodes start by advertising their presence, and when they learn about other nodes, spread their knowledge. That is, a node X knowing a route towards some other node advertises it to its neighbor, essentially meaning "*I can relay towards this end-receiver*", even if X may not know the actual identity of this end-receiver. Such an advertisement is hereby called a route proposal (one route is advertised at a time), and each node begins by

self-proposing. At the outcome of the topology dissemination, nodes may have several routes towards the same end-receiver, and thus several routing table entries for a same pseudonym $PS_{X\to R}$. Additionally, at network initialisation, each pair of neighbor nodes perform a DHKA and derives keying material to later encrypt circuit identifier and diverse routing information.

During a route proposal, several tasks are carried out: (i) the exchange of a circuit identifier, (ii) the communication of the adequate encryption of one $c_{\texttt{one}X\to R}$ used to encrypt payloads, and (iii) the computation of the pseudonym $PS_{X\to R}$, used by X to designate end-receiver R. This pseudonym computation actually consists in a two-round (three-message) exchange, where X and R run a secure multi-party computation (SMPC). Each node X may encounter several route proposals for the same end-receiver R, and must always obtain the same pseudonym $PS_{X\to R}$. Thus, pseudonyms can not be simply generated and handed over by the end-receiver R. Instead, the value of $PS_{X\to R}$ is determined by a secret of X, src_X, and a secret of R, dst_R, respectively representing the identity of X as end-sender, and that of R as end-receiver.

Because the end-receiver must be involved in pseudonym computations, messages must make a *return trip* between X to R for a route proposal to be completed. This is similar to the telescopic construction of circuits in Tor, with the difference that the construction starts from the end-receiver. This is depicted in Fig. 4.1, where R first self-proposes to X (solid lines in the figure), requiring communications only between X and R since they are direct neighbors in the topology graph. Then, X proposes this route towards R to Y (dashed lines in the figure), and must act as an intermediary node helping Y and R compute $PS_{Y\to R}$. This process goes on, effectively using the already existing routes to extend them by one more hop. Note that, in the example, payload messages will eventually then flow from Y towards R, *i.e.* in the opposite direction compared to the propagation of the route proposals.

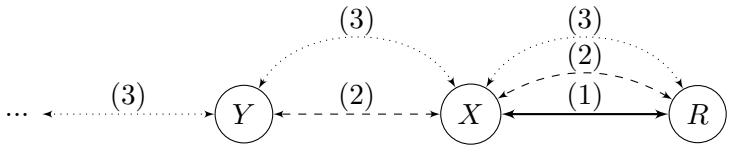

Figure 4.1. – Propagation of Route Proposals Relating to End-Receiver R

The circuits are built so that each node only knows its previous and next hop: even the end-receiver does not know the sequence of nodes constituting the circuit, nor its length. More generally, to ultimately prevent the adversary from breaking SA and RA, the topology dissemination phase is made as oblivious as possible, in order to conceal the constructed routes. For that, route proposals fulfill several security properties. In particular, self-proposals and relayed ones are indistinguishable. Also, routing messages involved in route proposals are actually mixed and re-ordered along with payload and dummy messages, preventing in particular the tracing of the return trip during the computation of pseudonyms.

This concludes the overview of the protocol. In the remainder of this chapter, the

complete protocol presentation builds on this overview. It roughly follows the same outline (routing tables, messages processing, and construction of routes), with the exception that oriented communications are presented last, once all the other components of the protocol have been described.

4.2. Routes and Routing Tables

The present section first reviews the keying material shared among neighboring pair of nodes. In a second time, it presents the contents of the routing table of a node in small example network.

4.2.1. Neighborhood Management

Since the protocol works over the Internet, any pair of nodes can theoretically be neighbors, since any node can directly communicate with any other (specific restrictions due to *e.g.* NAT set aside). However, we deliberately restrict the direct communication partners of nodes, and assume that the underlying topology graph is connected but incomplete. More exactly, the topology graph is considered *sparse*, in the terminology of Danezis [Dan03a], which means that nodes only have a few neighbors compared to the total number of nodes in the network. There are two main reasons for this. Similarly to Tarzan, the protocol uses dummy messages to conceal sending and receiving behavior. But using dummy messages on all n^2 links of the complete graph would be too costly. Secondly, this avoids nodes to disclose their IP addresses and their participation in the network to all other nodes, as noted by Clarke *et al.* [Cla+10]. Indeed, in a protocol where all nodes learn about all others, it is extremely easy to check the participation of a particular individual in the network: it is sufficient to run a node. In our case, each node only learns about the presence of a few (*e.g.* $\log n$) other nodes. Similarly as in a variant of Freenet [Cla+10], a node can for instance only connect to trusted nodes, realising a *friend-to-friend* network, and thus protecting the node's real-world identity. How these neighbors are selected in practice is out of the scope of this thesis.

Note that the attack against Tarzan, which relies on the fact that a node only builds circuits using the few other nodes it knew, does not apply here. Indeed, in the present protocol, nodes do not select their routes, they are constructed by the network as a whole.

At the start of the network, each node performs a DHKA with each of its neighbors. This assumes that all nodes agree on a specific group $\mathbb{G}$. For simplicity, we assume that the description of $\mathbb{G}$ (*i.e* the terms q and g) are publicly known. From the shared secret, they derive cryptographic materials. Namely, using a secure KDF, nodes generate two symmetric keys k_{XY_i} and k_{Y_iX} (one for each direction on the link) suitable for a block cipher such as AES, along with the necessary IVs. During the network lifetime, neighbors then perform a new DHKA periodically, and generate new keying material. This prevents the long-term compromise of link keys.

4.2.2. Routing Tables

During topology dissemination, a node X (not necessarily neighbor to node R) generally receives and possibly accepts several route proposals towards each node R. It also proposes several routes towards R as well. The nodes' routing tables at the outcome of topology dissemination differ depending on which node proposed, accepted, or refused which route proposals: the route proposal mechanism is probabilistic (see Section 4.5.4). A concrete example of constructed routes is depicted in Fig. 4.2 and Fig 4.3. The first figure depicts an example network centered around a node X and its neighbors Y_i, and the second shows the routing tables entries of node X, regarding end-receiver R only. There are two entries, meaning that X has two different routes towards R. In Fig. 4.2, the routes are depicted along with the circuit identifiers. The plain arrows correspond to the first route, and the dashed ones to the second. Note that the routes shown are only those towards R, and represent one configuration among many possible outcomes of topology dissemination in the example network.

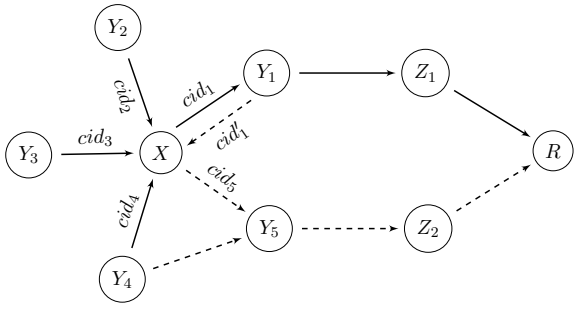

Figure 4.2. – Example Network

Prev. hop	PS	$c_{\texttt{one}}$	$c_{\texttt{prop}}$	Next Hop
Y_2, cid_2 Y_3, cid_3 Y_4, cid_4	$PS_{X\to R}$	$\mathsf{Enc}(pk_{Y_1,Z_1,R}, 1)$	$\mathsf{Enc}(pk_{Y_1,Z_1,R}, dst_R)$	Y_1, cid_1
Y_1, cid'_1	$PS_{X\to R}$	$\mathsf{Enc}(pk_{Y_5,Z_2,R}, 1)$	$\mathsf{Enc}(pk_{Y_5,Z_2,R}, dst_R)$	Y_5, cid_5

Figure 4.3. – Routing Table of Node X towards R

This example actually contains a lot of information. We will first look at each term in X's routing table, then analyse the depicted routes. The next subsection shows how tables are used for sending, relaying and receiving messages.

4.2.2.a) Terms in the Routing Tables

The *previous hop* and *next hop* fields are quite self-explanatory. The nodes are denoted with capital letters, *e.g.* Y, and can be understood as their IP address or their real-world

identity (since the two are assumed publicly linked in Assumption 1). When X receives a link message with a circuit identifier present in a *previous hop* field, it must forward accordingly to the node and with the circuit identifier of the corresponding *next hop* field. There can be several previous nodes (and circuit identifiers) for a given routing table entry, but there can be only one next hop. That is, forwarding is deterministic: once a message is sent on a route, the path it is going to take in the underlying topology graph is fully determined. This avoid issues with routing loops, as described in the next paragraph. The PS field contains the pseudonym used by X to designate end-receivers. It is used along with the *next hop* field in its activities as an end-sender. The true utility of pseudonyms appears later, during the realization of *oriented communications* (see Section 4.6). The $c_{\texttt{one}}$ field is the encryption of one used to encrypt payload messages that X sends, and to re-encrypt the ones X relays. It is encrypted under a product of public keys, where $pk_{Z_1,\ldots,Z_n}$ is a shorthand for $pk_{Z_1} \cdot pk_{Z_2} \cdot \ldots \cdot pk_{Z_n}$. The nodes $Z_1, Z_2, \ldots, Z_n$ are the nodes on the route between X and R. The first of the two entries, for instance, relates to a route going through Y_1, Z_1 and R, and thus $c_{\texttt{one}}$ is encrypted under $pk_{Y_1} \cdot pk_{Z_1} \cdot pk_R$. Lastly, the $c_{\texttt{prop}}$ field stores a ciphertext only used during the topology dissemination, for X to make route proposals towards R. It encrypts dst_R, a value secret to R and used to compute pseudonyms that nodes use to designate R.

Note that, even though the routing table entries in the example relate to end-receiver R, the IP address or identity of R never appears in them. Actually, X does not know that these entries relate to R.

4.2.2.b) Routes Depicted in The Example

There are two main routes in the example. The one corresponding to the first routing table entry is drawn with plain arrows, the other with dashed ones. The route relating to the first entry of X's routing table starts with either Y_2, Y_3, or Y_4, goes through X, and then $Y1$, Z_1 and finally arrives to R. The second one starts with Y_1, goes through X, Y_5, Z_2 and finally R. Additionally, a third route is depicted, showing that Y_4 also has a second route that directly reaches R through Y_5 and Z_2.

Circuits are *unidirectional*, meaning that payload messages are only meant to flow from X to R (but as we will see, it is necessary to let some routing messages go up the circuits). Circuits are also shared by nodes that compose them. Indeed, X may use its first routing table entry, and send its own messages to R through Y_1 and cid_1. But this link $X \xrightarrow{cid_1} Y_1$ may also be used by X to forward messages that *e.g.* Y_3 sends to R *via* X. Consequently, Y_1 can not know if a message on this link, with this circuit identifier, comes from X or from upstream nodes on the circuit (here, Y_2, Y_3, or Y_4). Abstractly, this participates in achieving SA and MU-session: circuits do not relate to one specific end-sender (and thus do not relate to any end-sender-receiver pair neither). In addition, nodes do not know who is part of the circuit, apart from their previous and next hops.

In the example network, X is not the only node with several routes towards R. Y_4 can reach R through sequence X-Y_1-Z_1, or through Y_5-Z_2. And Y_1 has a direct route *via* Z_1, but also an indirect one *via* X, Y_5 and Z_2. This shows that constructed routes may not always be the shortest ones. Also, in the example, X can reach R through Y_1 with cid_1,

and *vice-versa* (with cid'_1). This kind of situation is allowed in the network, and does not create routing loops, since circuits are separated and unidirectional.

4.3. Sending, Relaying, and Receiving Messages

With their routing tables, nodes have all the elements to send and relay messages. This section describes how nodes process messages from a *cryptographic* point of view, whereas the message re-ordering and use of dummy messages is the focus of the next one.

4.3.1. Link Message Format

The packet format of a link message carrying a payload message m for end-receiver R and meant to go through relay nodes $Z_1, Z_2, \ldots, Z_n$ is as follows:

$$\langle \{\texttt{payload} \| cid\}_k, \mathsf{Enc}(pk_{Z_1,\ldots,Z_n,R}, m_1), \mathsf{Enc}(pk_{Z_1,\ldots,Z_n,R}, m_2) \rangle \tag{4.1}$$

That is, a packet begins with a *header* consisting of an AES encryption of a circuit identifier accompanied by a `payload` flag, and two Elgamal ciphertexts. *All* link messages between any two nodes have this form, meaning each and every single message part of the protocol (including link messages that carry routing information). This means that all link messages have the same size and random-looking appearance (allowing to later *batch* them and shuffle them together). The flag here has value `payload`, but is also used to signal routing information or dummy messages. The header is constructed to always be $\lambda = 128$ bits long, *i.e.* one AES block. The symmetric key k used to encrypt the header is the link key for the direction of the link message (*e.g.* k_{XY} for a link message sent by X to its neighbor Y). Every message contains *two* Elgamal ciphertexts. Indeed, on several occasions, a second ciphertext is needed: for some routing messages, the second ciphertext is an encryption of one to perform (universal) re-encryptions; and during oriented communications, the first ciphertext contains the session identifier (that allows to link all payload messages in the session). If there are cases where a second ciphertext is not needed, an encryption of random data should still be included.

It is known that the Elgamal scheme can only encrypt group elements, *i.e.* elements from $\mathbb{G}$. To encrypt a piece of data $m \in \{0,1\}^*$ with the Elgamal scheme, an *encoding* is necessary to transform m into a group element. Yet, this encoding is known to degrade (or even take away) the homomorphic property of the scheme. In this work, this is not an issue: the homomorphic properties of the scheme are necessary only for *routing messages*. Thus, the protocol is built so that all plaintexts involved in routing messages are directly taken in $\mathbb{G}$. More information on this matter can be found in Appendix A.

4.3.2. Creating and Processing a Message

This section describes how a node encrypts a given payload message, and how it is then processed and relayed to its end-receiver. For that, we define the following primitives, for

$c = \mathsf{Enc}(pk, m, r)$ and $c_{\text{one}} = \mathsf{Enc}(pk, 1, r_{\text{one}})$:

$$\begin{aligned}
\mathsf{ReEnc}_{\mathsf{one}}(c_{\text{one}}) &\coloneqq \mathsf{ScExp}(c_{\text{one}}, r') \text{ with } r' \leftarrow_{\$} \mathbb{Z}_q \\
&= ({c_{\text{one}}}_0^{r'}, {c_{\text{one}}}_1^{r'}) \\
&= \mathsf{Enc}(pk, 1, r_{\text{one}} \cdot r') \\
\mathsf{ReEnc}_{\mathsf{nopk}}(c_{\text{one}}, c) &\coloneqq \mathsf{CtxtMult}(c, \mathsf{ReEnc}_{\mathsf{one}}(c_{\text{one}})) \\
&= (c_0 \cdot {c_{\text{one}}}_0^{r'}, c_1 \cdot {c_{\text{one}}}_1^{r'}) \\
&= \mathsf{Enc}(pk, m, r + r_{\text{one}} \cdot r') \\
\mathsf{Enc}_{\mathsf{nopk}}(c_{\text{one}}, m) &\coloneqq \mathsf{PlainMult}(\mathsf{ReEnc}_{\mathsf{one}}(c_{\text{one}}), m) \\
&= (g^{r_{\text{one}} \cdot r'}, m \cdot h^{r_{\text{one}} \cdot r'}) \\
&= \mathsf{Enc}(pk, m, r_{\text{one}} \cdot r')
\end{aligned}$$

The $\mathsf{Enc}_{\mathsf{nopk}}$ operation leverages the $\mathsf{PlainMult}$ homomorphic operation of the Elgamal scheme, and allows a node to encrypt a plaintext using an encryption of one, and without a public key. The $\mathsf{ReEnc}_{\mathsf{one}}$ and $\mathsf{ReEnc}_{\mathsf{nopk}}$ are actually the atomic operations realised in the UReEnc primitive. Indeed, the latter can be expressed as:

$$\mathsf{UReEnc}(C = (c, c_{\text{one}})) = (\mathsf{ReEnc}_{\mathsf{nopk}}(c_{\text{one}}, c), \mathsf{ReEnc}_{\mathsf{one}}(c_{\text{one}})) \tag{4.2}$$

In contrast with other protocols that use URE, the UReEnc primitive is split . Indeed, in the present protocol the topology dissemination phase allows to distribute the encryptions of one. As a result, in the protocol, an Elgamal ciphertext does not always need to be accompanied with an encryption of one (as it would be in the URE-Elgamal scheme). In addition to saving bandwidth, this optimisation also allows nodes to generate many re-encryptions of one in an asynchronous and pre-emptive manner, so that they need only to perform one $\mathsf{CtxtMult}$ operation during the actual forwarding of messages.

With these primitives, given a payload message $m = m_1 \| m_2$ and a end-receiver's pseudonym $PS_{X \to R}$, X proceeds in the following way to send m in the example network of Fig 4.2. First, it selects one of its two routing table entries for $PS_{X \to R}$ at random. Let's assume that X chooses its second routing table entry. For this case, Fig. 4.4 describes the sequence of link messages from X to R. The involved Elgamal ciphertexts are processed as follows. First, X gets the ciphertext ${c_{\text{one}}}_{X \to R} = \mathsf{Enc}(pk_{Y_5, Z_2, R}, 1)$ from its routing table entry, and computes:

$$\begin{aligned}
c_1 &\leftarrow \mathsf{Enc}_{\mathsf{nopk}}({c_{\text{one}}}_{X \to R}, m_1) \\
c_2 &\leftarrow \mathsf{Enc}_{\mathsf{nopk}}({c_{\text{one}}}_{X \to R}, m_2)
\end{aligned}$$

X sends these ciphertexts to Y_5. Upon receiving them, Y_5 finds ${c_{\text{one}}}_{Y_5 \to R} = \mathsf{Enc}(pk_{Z_2, R}, 1)$ by a table lookup on previous hop (X, cid_5). It then partially decrypts c_1 and c_2, and re-encrypts them:

$$\begin{aligned}
c_1' &\leftarrow \mathsf{ReEnc}_{\mathsf{nopk}}({c_{\text{one}}}_{Y_5 \to R}, \mathsf{Dec}(sk_{Y_5}, c_1)) \\
c_2' &\leftarrow \mathsf{ReEnc}_{\mathsf{nopk}}({c_{\text{one}}}_{Y_5 \to R}, \mathsf{Dec}(sk_{Y_5}, c_2))
\end{aligned}$$

Y_5 then forwards the ciphertexts to Z_2, and the process repeats until it reaches R. The partial decryption carried out by relay nodes can clearly be seen, ensuring that at the last hop, the Elgamal ciphertext are simply encrypted under R's public key. This partial decryption and re-encryption of relay nodes is very similar to the processing that Huang *et al.* [HLF12] apply in their own protocol.

$$
\begin{array}{lll}
X \to Y_5 & : \{\texttt{payload} \| cid_5\}_{k_{XY_5}}, & \underbrace{\mathsf{Enc}(pk_{Y_5,Z_2,R}, m_1)}_{c_1}, \quad \underbrace{\mathsf{Enc}(pk_{Y_5,Z_2,R}, m_2)}_{c_2} \\
Y_5 \to Z_2 & : \{\texttt{payload} \| cid' \}_{k_{Y_5Z_2}}, & \underbrace{\mathsf{Enc}(pk_{Z_2,R}, m_1)}_{c'_1}, \quad \underbrace{\mathsf{Enc}(pk_{Z_2,R}, m_2)}_{c'_2} \\
Z_2 \to R & : \{\texttt{payload} \| cid'' \}_{k_{Z_2R}}, & \mathsf{Enc}(pk_R, m_1), \quad \mathsf{Enc}(pk_R, m_2)
\end{array}
$$

Figure 4.4. – Sequence of Link Messages from X to R

In terms of security, the above processing of messages participates in ensuring MU-tracing (and RA to some extent, since R's public key is never used). First note that, although it does not appear explicitly, the management of the IVs for the AES encryptions ensures that all link messages between neighboring nodes always exhibit a different, random-looking header, by using a different IV for each message. Then, for the Elgamal ciphertexts, note that care is taken to re-encrypt the c_{one} ciphertext before *every* use, thus effectively emulating the UReEnc primitive, and allowing to re-use Golle *et al.*'s USS security property. This intuitively means that ciphertexts change at each hop in such a way that they are not recognisable, even by the end-sender X. Then, because plaintexts are encrypted under a *product* of public key (similarly to Huang *et al.* [HLF12]), rather than solely under pk_R, the protocol resists the re-encryption specific attack described in Section 3.4.1. Finally, by the probabilistic nature of the Elgamal scheme, there are many encryptions of one for a given (product of) public key(s); and because Elgamal ciphertexts do not leak which key(s) they are encrypted under since the scheme ensures key-privacy, encryptions of one do not act as a network-wide identifier.

4.4. Messages Re-Ordering, Dummy Messages, and Controlled Traffic Rates

The previous section only presents the cryptographic processing of messages, and how to ensure MU. This section is complementary: it presents countermeasures against network-level attacks, ultimately seeking to ensure TAR, SA, and RA. This is performed by a conjunction of three tools: message-reordering, dummy messages, and controlled traffic rates.

These three tools are interdependent, and must be presented together. In particular, as noted by Diaz and Preneel [DP04], the policy for producing and emitting dummy messages has to be designed in conjunction with the message re-ordering mechanism.

Moreover, as in the Tarzan protocol, the controlled traffic rate mechanism goes hand in hand with the production of dummy messages as cover traffic for a node's neighbors. Here, message re-ordering is performed using a variant of the *timed dynamic pool* strategy used by the Mixmaster protocol [Mol+03], but adapted to the homogeneous, fully distributed setting. On the other hand, dummy messages and controlled rates are inspired from the Tarzan protocol, but are adapted to provide more robust security guarantees.

This section first presents a thorough analysis leading to strict rules that a node must follow to ensure SA and RA, using dummy messages and controlled traffic rates. In a second time, we present how to implement these rules in accordance with the message re-ordering mechanism.

To avoid confusion, the following definition formally makes the distinction between real and dummy messages. Recall also Definition 8 which makes the difference between sender and end-sender (and likewise for receivers). These terms are used extensively in what follows.

Definition 16 (Dummy and Real Messages). *A dummy message is a link message carrying no payload nor routing information, and not meant to be relayed further than the link on which it is sent. A real message is defined in opposition, as a link message carrying a payload or routing message which is meant for a specific end-receiver.*

A dummy message from node X to its neighbor Y consists in the following, for $r_1, r_2 \leftarrow_{\$} \mathbb{G}$:

$$\left\langle \{\texttt{dummy}\}_{k_{XY}}, \mathsf{Enc}(pk_X, r_1), \mathsf{Enc}(pk_X, r_2) \right\rangle$$

4.4.1. Dummy Messages and Controlled Traffic Rates for SA and RA

One of the features ensuring the security of the protocol is that nodes conceal their own traffic (messages they end-send or end-receive) within the traffic of their neighbors. This realises a homogeneous architecture. Yet, a homogeneous architecture is not enough. First, because the protocol does not make any assumption on the traffic load in the network, it is possible that a node X does not get any traffic from its neighbors for a certain period of time, and thus no cover traffic. Then, even if there is a lot of traffic passing through X, a network observer can still count its incoming and outgoing messages, and breach SA and/or RA, as exposed in Section 3.4.4.

This section shows how to ensure SA and RA, by preventing the observation of end-sending and end-receiving activities. For that, we present an analysis of the threats posed by network observers and by (collusions of) corrupted neighbors, and how dummy messages and controlled traffic rates can protect against them. These two kinds of adversaries call for different approaches: a global observer sees all links a node has with its neighbors, but can not distinguish a dummy message from a real one; while corrupted neighbors do not see all the links of a node, but they detect dummy messages that it sends to and receives from them. This section progresses in incremental steps, building up towards a solution that ensures SA and RA against both types of adversaries at the same time.

4.4.1.a) A First Step With Network Observers

We first assume that the adversary is only a global network observer. For a given target node X, this means that every incoming and outgoing link message of X, to or from any neighbor, is visible to the adversary (even though they are protected by encryption). It is assumed that end-sending is detected by the adversary when the number of (real) incoming messages of X, noted I, is lower than the number of its outgoing messages O. Conversely, end-receiving is detected when $I > O$. The goal is thus that, from the adversary's point of view, I *seems to be equal* to O, *i.e.* to make X appear as a simple relay pipe.

First of all, note that, without any particular addition to the protocol, I and O can be made equal, if X end-sends a message when and only when it end-receives one. This however puts extreme constraints on the nodes and on the network as a whole, and is highly impractical.

To provide more flexibility, dummy messages are used. By the properties of the AES and Elgamal schemes, observers can not distinguish between dummy and real messages. Thus, to *perturb* their observations of the numbers I and O, nodes can send dummy messages. But there must be a specific strategy in the sending of these dummy messages, a *policy* that makes I and O appear equal. In particular, simple strategies such as randomly sending dummy messages from time to time, or sending one every t seconds exactly, are not so useful. Indeed, a basic statistical analysis can be enough for the observer to work around such simple policies. Consequently, the dummy message policy must instead depend on past *and future* values of I and O.

Hereby, in accordance with the message re-ordering mechanism presented later, we divide time into discrete time intervals, corresponding to the batching *rounds*. The goal of the dummy message policy thus becomes that, in each round and for each node, $I = O$ from the point of view of the adversary. Against mere network observers, and without taking corrupted neighbors into account, this means that in round r: (i) a node can end-send only if it received at least one dummy message from one neighbor in the same round, and (ii) a node must send one dummy message to one neighbor when it end-receives in round r. In point (i), the *controlled traffic rates* mechanism begins to appear: a node must *retain* messages it wants to end-send *until* some condition is satisfied. To be complete, the policy should include a third point, briefly mentioned earlier: (iii) a node can also end-send if it end-received in the same round.

There is one pitfall that immediately poses a problem in this basic policy. An *initialisation issue* arises: which node X will send the first dummy message(s), that will allow its neighbor(s) to end-send? More generally, how to prevent the network from *stalling* because no node emits dummy messages to any neighbor? This means that the policy must ensure that each node provides enough dummy messages to its neighbors. This idea is included in our final solution, built up in subsequent sections. Before presenting it, however, the next section considers the impact of corrupted neighbors on the observations and comments made so far.

4.4.1.b) Considering Corrupted Neighbors

Against corrupted neighbors, dummy messages have a limited impact, since those can trivially differentiate them from real messages. In the worst case, when a node is surrounded by a collusion of corrupted neighbors, dummy messages are entirely useless, and the only strategy that makes I and O look equal to the adversary is point (iii) of the above policy: X *systematically* end-sends a message when, *and only when*, it end-receives one.

In addition, when considering corrupted neighbors, the policy proposed above may actually reveal more information than no policy at all. Indeed, if X follows it to the letter and sends dummy messages only on a *need basis*, corrupted neighbors obtain a easy way of detecting when a node end-receives. Indeed, a node that end-receives can be distinguished from one that simply relays a message based on the fact that the former one is the only one sending out dummy messages. This calls for a more complex policy, in particular where the emission of dummy messages does not depend on the actual end-sent or end-received messages *from the point of view of the adversary*.

Since the issue seems to be that X can be detected as end-receiver *if it sends a dummy message to a corrupted neighbor*, a naïve approach would be to consider that X has at least one honest neighbor, and reformulate points (i) and (ii) of basic policy as: (i) X can end-send only if it received at least one dummy message *from an honest neighbor*, and (ii) X must send one dummy message to an honest neighbor when it end-receives. This works, because dummy messages exchanged between X and its honest neighbors are indistinguishable from real ones, even for corrupted neighbors. In a sense X's honest neighbors act as a *relief valve* where it gets or dumps cover traffic. This is the reason why we make Assumption 3, that a node has at least one honest neighbor.

But things are not that simple: X does not know which of its neighbors are honest. Assumption 3, only states that *one* neighbor is honest, but not which one. This means that X has to assume simultaneously that each node may be corrupt. More exactly, a more conservative version of Assumption 3 is to consider that the neighborhood of X is partitioned in a least two collusions that do not share knowledge between them. This allows X to use dummy messages sent to or received from one collusion to fool the other. And since X is not aware of this partition, it must assume that every possible partition holds simultaneously, and use a policy that ensures SA and RA in every case[1].

In what follows, we formalise the problem and build up towards an applicable solution, that each node can individually apply.

4.4.1.c) With Formalism and Known Honest Neighbors

The neighbor set of X is denoted $\mathfrak{n}$. In a first time, we assume that X knows that $\mathfrak{n}$ is partitioned in two subsets: honest neighbors $\mathfrak{h} \subseteq \mathfrak{n}$, and corrupted neighbors $\mathfrak{c} \subset \mathfrak{n}$. The corrupted nodes are assumed to collude together and share their observations and

[1] Alternatively, to actually increase its chances of *not* being surrounded by one big collusion of corrupted nodes and realise Assumption 3, X can mount a *reverse Sybil attack*, *i.e.* run several nodes with different identities, and connect with them in the underlying topology graph.

knowledge together and with a global network observer. First, the term I is refined according to the nature of the messages (real or dummy), and to their provenance (honest or corrupted neighbors). That is:

$$I = I_{dum} + I_{real} = I^{\mathfrak{h}} + I^{\mathfrak{c}}$$

Additionally, to denote, say, dummy messages received from honest neighbors, the term $I^{\mathfrak{h}}_{dum}$ is used, combining subscript and superscript notations. The same notations are defined for outgoing messages O.

Ultimately, the aim is to make nodes in $\mathfrak{c}$ and the global observer *believe* that $O_{real} = I_{real}$, even though X does end-send and end-receive messages. This formally translates into the following equations to be respected in each round r. They respectively correspond to points (i) and (ii) expressed in the previous section. Point (iii) also appears implicitly.

$$O^{\mathfrak{c}}_{real} \leq I_{real} + I^{\mathfrak{h}}_{dum} \tag{4.3}$$

$$I^{\mathfrak{c}}_{real} \leq O_{real} + O^{\mathfrak{h}}_{dum} \tag{4.4}$$

Put together, these relations are equivalent to eq. (4.5), showing that node X can adjust the difference between its outgoing and incoming packets in the margin provided by the traffic to and from its honest neighbors.

$$-O^{\mathfrak{h}} \leq O^{\mathfrak{c}}_{real} - I^{\mathfrak{c}}_{real} \leq I^{\mathfrak{h}} \tag{4.5}$$

4.4.1.d) With Formalism and Known Neighbor Collusions

A conclusion from this first naïve analysis is that X can end-send and end-receive as many real messages as it wants through its honest neighbors. However, two issues arise: a heavy use of honest neighbors as *relief valve* will be detected, and, anyway, X does not know which of its neighbors are honest and can not rely on this strategy. To make a step towards lifting assumption that X knows which neighbors are honest and which are corrupted, we model the neighborhood of X as a partition of collusions $\mathfrak{C} = \{\mathfrak{c}_1, \mathfrak{c}_2, \dots\}$, such that $\cup_i \mathfrak{c}_i = \mathfrak{n}$ and $\mathfrak{c}_i \cap \mathfrak{c}_j = \emptyset$ for $i \neq j$. That is, there are no *honest* nodes; only groups of corrupted nodes sharing knowledge within their collusion, but not with other collusions. Now, if X knows the partition $\mathfrak{C}$ it can go around the collusions, using the dummy messages sent to and received from one collusion to fool the others. For that, there must be at least two collusions in the partition $\mathfrak{C}$. This can be seen as a variant of Assumption 3. In this setting, eq. (4.5) must hold for each collusion in $\mathfrak{C}$:

$$\forall\ \mathfrak{c}_i \in \mathfrak{C}:\ -O^{\mathfrak{n}\setminus\mathfrak{c}_i} \leq O^{\mathfrak{c}_i}_{real} - I^{\mathfrak{c}_i}_{real} \leq I^{\mathfrak{n}\setminus\mathfrak{c}_i} \tag{4.6}$$

4.4.1.e) With Formalism and Unknown Neighborhood

Finally, let us lift the assumption that X knows the collusion partition of its neighborhood. The only safe option for X is to act as though *all* possible collusion partitions were *simultaneously* in effect. That is, it must enforce eq. (4.6) for each possible partition.

However, if X has k neighbors, there are $\sum_{i\in[2,k]} i \cdot S(k,i)$ such equations to enforce, where $S(k,i)$ are the Stirling set numbers. Indeed, for each possible partition $\mathfrak{C}$ containing i collusions, there are i equations to respect. And by definition, there are $S(k,i)$ partitions of size i. This is clearly an impractical policy to implement: heuristics must be used. The first simplification that can be made is to take into account only the *worst case* collusion partitions. That is, partitions of size two, where all but one neighbor are in the same collusion. There are k such partitions, one per neighbor, and X must thus respect $2k$ equations. This is a great improvement, but still impractical to enforce. To further simplify the problem, we examine these equations in the case of SA.

Example 1 (Example for SA and $k = 3$ neighbors). *Let X be a node with $k = 3$ neighbors Y_1, Y_2, and Y_3. To ensure SA (only), X must verify the following 6 equations derived from eq. (4.3) corresponding to SA:*

$$\begin{aligned}
&\text{For } \mathfrak{C}_1 = \{\{Y_1\},\{Y_2,Y_3\}\} : && O_{real}^{Y_2,Y_3} \leq I_{real} + I_{dum}^{Y_1} && O_{real}^{Y_1} \leq I_{real} + I_{dum}^{Y_2,Y_3}\\
&\text{For } \mathfrak{C}_2 = \{\{Y_2\},\{Y_1,Y_3\}\} : && O_{real}^{Y_1,Y_3} \leq I_{real} + I_{dum}^{Y_2} && O_{real}^{Y_2} \leq I_{real} + I_{dum}^{Y_1,Y_3}\\
&\text{For } \mathfrak{C}_3 = \{\{Y_3\},\{Y_1,Y_2\}\} : && O_{real}^{Y_1,Y_2} \leq I_{real} + I_{dum}^{Y_3} && O_{real}^{Y_3} \leq I_{real} + I_{dum}^{Y_1,Y_2}
\end{aligned}$$

To simplify the system, it is possible to replace all the I_{dum} terms with the minimum of all of them, denoted $min(I_{dum})$, without violating any of the relations. A value $min(I_{dum}) = n$ means that, in the current round, X received at least n dummy messages from *each* neighbor. Secondly, since $O_{real} = O_{real}^{Y_1} + O_{real}^{Y_2} + O_{real}^{Y_3}$, X can simply ensure that $O_{real} \leq I_{real} + min(I_{dum})$. We successfully reduced the system to one simple constraint to ensure SA. Proceeding in the same manner for RA shows that it is sufficient for X to ensure $O_{real} + min(O_{dum}) \geq I_{real}$.

In definitive, in order to guarantee both SA and RA without any assumption on the neighborhood other than Assumption 3, each node must ensure that:

$$-\,min(O_{dum}) \leq O_{real} - I_{real} \leq min(I_{dum}) \tag{4.7}$$

Taking a step back, the *traffic rates equation* (4.7) says that X must neither *send* nor *receive too many real messages*. Here, "real messages" designate both relayed and end-sent/end-received messages. More specifically, each node must maintain an *equilibrium* between link-sent and link-received real messages: in a given round, it must link-send approximately as many real messages as it link-received, up to the bounds provided by its link-sent and link-received dummy messages in this same round. We call these bounds the *dummy budgets*. The *sending dummy budget* is the lower bound in eq. (4.7), equal to the *minimum* over the number of dummy messages that the node *received* from each neighbor in that round. It is the *minimum* over all neighbors that is considered, because, to protect from every neighbor, X has to adapt to the margins (in terms of dummy messages) given by the most restricting neighbor. The *receiving dummy budget* is defined in a similar manner, according to the minimum over all neighbors of *sent* dummy messages.

Note that eq.(4.7) implies that if X is a simple relay node (neither end-sending nor end-receiving), it does not need to send nor receive dummy messages. Yet, as explained

earlier, nodes should not send dummy messages *only* on a need basis. More generally, the dummy message and controlled traffic rate policy must be implemented to respect eq. (4.7), while keeping in mind all that has been learned in our analysis. We show next how a node can proceed to enforce the equation in practice.

4.4.2. Integration With Pool-Based Batching

This section shows how the dummy messages and the controlled traffic rates policies can be implemented in conjunction with the message re-ordering mechanism.

4.4.2.a) Batching With Timed Dynamic Pools

Existing literature indicates that pool-based mixes are the most resistant to traffic analysis (see Section 3.4) [BPS01; SDS02]. We choose to use the already well-tested timed dynamic pool of Mixmaster [Mol+03], also known as a *Cottrell mix* [SDS02]. A node implementing this batching strategy places all link messages that need to be sent (*i.e.* relayed or end-sent) in a pool $\mathcal{P}$. The system is parameterized by a time interval $t_{\mathcal{P}}$, a minimal number of messages in the pool $n_{\mathcal{P}min}$, and a fraction $f_{\mathcal{P}}$. Every $t_{\mathcal{P}}$ seconds, if there are $n_{\mathcal{P}}$ messages in the pool, the node randomly selects and sends $n \coloneqq \min(n_{\mathcal{P}} - n_{\mathcal{P}min}, n_{\mathcal{P}} \cdot f_{\mathcal{P}})$ messages from the pool.

In this thesis, we propose to use a variant of this mechanism, in which a node maintains one separate pool per neighbor. This facilitates the implementation of the dummy message policy described in the previous section (which needs to have a per-neighbor control on dummy messages). Also, this design choice allows to ensure that, in every round, each neighbor gets the same number of link messages (regardless of whether they are real or dummy ones). This prevents against attacks on *asynchronous free-route* mixnets [BPS01] (the present network is indeed asynchronous, and not a cascade). The latter constraint can be formalised as $O^{Y_i} = O^{Y_j}$ for all neighbors Y_i, Y_j at each round r.

Therefore, each node X maintains one pool $\mathcal{P}_{Y_i}$ per neighbor Y_i. When X must send or relay a message to Y_i (be it a real or dummy one), it places it in $\mathcal{P}_{Y_i}$. Every $t_{\mathcal{P}}$ seconds, X sends out messages if *all* pools have enough messages in them. More exactly, let $n_{\mathcal{P}_{Y_i}}$ be the number of messages in pool $\mathcal{P}_{Y_i}$. At each round, X randomly picks n messages from each pool, and sends them in a random order, for n defined as:

$$n \coloneqq \min_i(\min(n_{\mathcal{P}_{Y_i}} - n_{\mathcal{P}min}, n_{\mathcal{P}_{Y_i}} \cdot f_{\mathcal{P}})) \tag{4.8}$$

4.4.2.b) Producing Dummy Messages

With this batching strategy, dummy messages are inserted into pools, according to the following policy. At the beginning of each round, X inserts a dummy message in a random fraction f_{dum} of neighbor pools (*e.g.* $f_{dum} = 1/3$). Additionally, rounds in which n is equal to zero according to eq. (4.8), X still sends one dummy message to each neighbor. This very basic policy has the advantage of respecting the lessons learned from our analysis. That is, dummy messages are not sent deterministically and on a *need basis*, since the policy is completely independent from the end-sent and end-received

messages. And secondly, the network is prevented from *stalling*, since each node regularly provides neighbors with dummy messages: each pool gets a dummy every $1/f_{dum}$ rounds on average, and when a node can not *fire* its pools, it still sends out dummies.

4.4.2.c) Traffic Rates Constraints in Practice

There is an apparent inconsistency between the batching strategy on the one hand, and the need to respect the traffic rate equation (4.7) on the other hand. Indeed, the latter dictates that messages should be randomly selected from pools, whilst the former requires that, at each round, the batch of selected messages fulfill certain constraints. To resolve this, we relax the constraints on the traffic rates, conserve the random sampling from pools, but apply a post-processing to the obtained batches.

In practice, a node X can ensure the traffic rate equation *at each round* in the following way. At the beginning of each round, X counts the number of real and dummy link messages received during the last round, and deduces I_{real}, and $I_{dum}^{Y_i}$ for each neighbor Y_i. It processes the messages according to the protocol, possibly placing in its neighbors' pools new real messages to end-send or relay. At the end of the round, from I_{real} and $min(I_{dum})$, X can deduce the set of solutions to the traffic rates equation, where O_{real} and $min(O_{dum})$ are considered as variables, denoted x and y. The set of solutions is $S = \{(x, y) \mid y \in [0, I_{real}], x \in [I_{real} - y, I_{real} + min(I_{dum})]\}$. Then, X samples a batch of messages from the neighbor pools. If the sampled batches contain a number of real and dummy messages that fit into S, then X can safely send them. Otherwise, we resort to a post-processing of the batches. We distinguish two cases: (i) either O_{real} is too high to fit into any of the solutions in S (meaning X is trying to send *too many* real messages), (ii) either O_{real} or $min(O_{dum})$ are too low.

In case (i), to decrease O_{real}, X chooses a real message at random from the batches, replaces it by a dummy message, and repeats the process until the batches fit into the solutions. In case (ii), the situation is not so simple. The simplest approach would be to either increase O_{real} by taking other real messages from the pools, or to increase $min(O_{dum})$ by adding in the batches one additional dummy message for each neighbor. The latter case is to be avoided, for a reason already discussed: a node that sends many (dummy) messages in a round is easily detected by the adversary as a node that (end-)receives many messages. We also reject the approach of augmenting O_{real} by manually selecting real messages in the pools, for several reasons: this strongly tampers with the probabilistic nature of the random batch sampling mechanism, it implies that a node can be easily forced to send out all its real messages stored in its pools (a neighbor only needs to send many real messages to it in one round), and it simply fails when there are no more real messages in the pools.

In order to address case (ii) properly, we choose to relax the traffic rates constraints on several rounds, and to use *end-to-end dummy messages* (*i.e.* `payload` messages that encrypt only an `e2e-dummy` flag) as a last resort. That is, to allow a node to handle a sudden surge in incoming traffic (a high I_{real} value), when case (ii) appears, a node is allowed *not* to respect the traffic rate equation straight away. Instead, it can postpone the *resolution* of these constraints to a latter round $r + \Delta r$, for some parameter Δr. For

that, it keeps track of unresolved constraints for the last Δr past rounds. In most cases, since all end-senders respect eq. (4.7) and thus send their messages at rather low rates, a high I_{real} value in round r is likely to be naturally *absorbed* over Δr rounds. When that is not the case, *i.e.* when the constraint from round $r - \Delta r$ is still unresolved, we resort to *end-to-end dummy messages*. That is, X replaces dummy messages in the batches with end-to-end dummy ones, that *look like* real messages to X's neighbors and more generally, to all nodes except its end-receiver. X sets the *cid* value of an end-to-end dummy message to that of a random next hop in its routing table, effectively meaning that X chooses a random end-receiver for that end-to-end dummy message.

4.4.2.d) Concluding Remarks

The proposed approach ensures TAR, SA, and RA in our adversary model. While message ordering alone is sufficient to ensure TAR, it is all three tools together (the dummy messages, the traffic rates, and the message re-ordering) that protect SA and RA. Indeed, the message re-ordering system, as proposed in this thesis, mixes together end-sent and relayed messages. It is the advantage obtained from the adaptation of mixnet techniques into a homogeneous network architecture, bringing uncertainty to the adversary when trying to perform advanced traffic analysis based on *flow fingerprints* (as described in Chapter 3). This element of design, to the best of our knowledge, is not present in the literature. In comparison, the Tarzan protocol [FM02] does not use message re-ordering. The thorough analysis conducted in Section 4.4.1 also shows that, contrarily to our protocol, Tarzan's dummy messages and traffic rates policy fails to ensure SA against a *collusion* of corrupted neighbors

However, the provided security comes at a great cost in terms of delivery latency. This cost is measured in Chapter 6. Also, as the authors of Tarzan note, having nodes wait for dummy messages from neighbors facilitates DoS attacks: it is sufficient for *one* corrupted neighbor of X to refrain from sending any dummy messages to greatly limit the end-sending rate of X. This is however an active attack, not included in the adversary model. It is also easily detectable, and the misbehaving node can be discarded from the network.

4.5. Constructing the Routes

So far, we have presented routing tables, how they are used, and how the message forwarding mechanism is designed. It remains to explain how the nodes actually acquire the necessary information to fill their routing table, such as the pseudonyms and the c_{one} ciphertexts. This section bridges that gap, by fully presenting the topology dissemination phase, in which nodes make *route proposals* to learn about each other, build circuits, and compute pseudonyms.

This section abstractly defines how topology dissemination is carried out in the proposed protocol. In a second time, it presents the construction of pseudonyms and the *route proposal* mechanism in detail. Finally, it discusses the *route proposal policy*, which

determines which routes are built among all the possible paths in the network, and according to which characteristics.

4.5.1. Ideas and Aim of Topology Dissemination

The topology dissemination in the proposed protocol is quite standard in essence, since it is essentially a *gossip* protocol, in which each node exchanges information about the network with its neighbors. It begins at network startup, where each node X has only a view of its direct neighborhood, and ends when each node learned about all other nodes in the network. More exactly, during topology dissemination, long-lived circuits are built starting from the end-receivers, and are extended towards the edges of the network. This methodology contrasts with the standard sender-initiated circuits commonly found in anonymous protocols such as Tor [DMS04] or Tarzan [FM02]. In the present protocol, we use static circuits shared by several nodes, *i.e.* each node in the circuit (except the end-receiver) is potentially an end-sender, and also a relay for the upstream nodes in the circuit. However, a given node X that is part of some circuit c does not know the identities of the other nodes that are in c, except for its previous and next hop (even if X is the end-receiver itself).

However, one main difference between this protocol and a standard network discovery protocol lies in the fact that nodes do not actually learn the IP address of nodes more than one hop away from them; instead, they learn a pseudonym. Also, for privacy to hold, most of the information usually exchanged during a standard network discovery protocol (such as the exact length of a route) is concealed to nodes. To compute pseudonyms, nodes share their knowledge of the network with other nodes only *one route at a time* (rather than exchanging *e.g.* a list of reachable IP addresses along with a hop count metric as in RIP). More exactly, we say that a node actually *proposes* to its neighbors to relay their messages towards some anonymous end-receiver, by extending an already existing circuit by one hop. Thus, *route proposals* are the atomic information exchange operation at the heart of the topology dissemination.

4.5.1.a) Definition of Route Proposals

The main goal of a route proposal is to *create and extend routes*, thus allowing nodes to learn about (the pseudonyms of) other nodes in the network, along with all the necessary information to end-send and relay payload messages towards them.

Definition 17 (Route Proposal, Proposer, Proposee). *Nodes build and extend routes* via route proposals. *The notation* $\texttt{RP}(X{\leftrightarrow}Y{\rightarrow}R)$ *denotes a* route proposal *by node* Y *(the* proposer*), to its neighbor* X *(the* proposee*), towards end-receiver* R*, meaning that* Y *offers* X *to relay its messages towards* R*.*

In practice, a route proposal consists in a short interaction between the proposer, the proposee, and the end-receiver, that consists in the exchange of *routing messages*. At the outcome of this interaction, the route is extended by one hop: Y notes in its routing tables that it must relay X's messages towards R (by adding X and a circuit identifier

cid as *previous hop* in the adequate entry); and X creates a new entry in its routing table, with Y and the same *cid* as next hop. Additionally, the proposee learns the pseudonym $PS_{X \to R}$ that it is going to use to designate the end-receiver (but not its identity or IP address), and an encryption of one, denoted $c_{\texttt{one}X \to R}$, under the appropriate public key(s) in order to encrypt and re-encrypt messages for R.

It is necessary to involve the end-receiver in the process of route proposals, in order to compute the pseudonym. Indeed, if a given proposee X gets two different route proposals towards the same end-receiver R in two different moments of the lifetime of the network, it *must* get the same pseudonym $PS_{X \to R}$. This allows X to know the number of routes it has towards the same end-receiver, and to later contact R as an end-sender. A consequence of the need to solicit the end-receiver in a route proposal is that routing messages must be relayed (back and forth) over the route between the proposer and the end-receiver: a *return trip* is necessary. This is performed by using circuits in a reverse fashion, similarly to how Tor routes the answers of receivers back to users. Note however that the proposer and proposee of any route proposal are always neighbors.

At the initialisation of the network, or upon joining it, each node knows only one route and one end-receiver: itself. Thus, the first action that a node R performs in the network is a *self-proposal*. After this self-proposal, one route is created between R and each of its neighbors (routes of length one). Then, its neighbors *relay* this proposal, thus extending the routes, and making R known (under its different pseudonyms) by more nodes. Then, the 2-hop neighbors of R relay it, etc. A route proposal is relayed in this way, and propagates from R to the edge of the network.

Definition 18 (Self-Proposal, Relayed Proposal). *A self-proposal* $\texttt{RP}(X \leftrightarrow R \to R)$ *is a* route proposal *in which the end-receiver is also the* proposer*. A relayed proposal is defined as any route proposal that is not a* self-proposal*.*

Of course, proposees are allowed to decide whether they *accept* or *refuse* route proposals, based on several characteristics (*e.g.* the number of routes it already knows). If they refuse it, the circuit is not extended. Similarly, when a proposee accepts a route proposal, it can choose to relay it to its own neighbors or not. These decisions of node X depend on various pieces of information, and are captured by the *route proposal policy*.

Definition 19 (Route Propocal Policy). *Proposees make the decision to* accept *or* refuse *a route proposal, and to relay it further or not, according to the route proposal policy. This term encompasses both the information that the proposee has access to, and the (probabilistic) decision process that takes place based on this information.*

It is the route proposal policy that determines when the *propagation* of route proposal stops, *i.e.* when the network reaches a stable state, and no more route are proposed. A sound route proposal policy must make sure that this happens only *after* all nodes learn about each other. The route proposal policy is discussed in more details in Section 4.5.4.

4.5.1.b) Privacy Properties of Route Proposal

Aside from these functional goals, the mechanism of route proposal must refrain from providing the adversary with elements allowing her to ultimately break SA, RA, or SU, or MU. More exactly, although these anonymity properties must be ensured for actual communications (*i.e.* for payload messages), it is still necessary to ensure some form of privacy for route proposals as well (*i.e.* for routing messages). Indeed, if the building of circuits was completely open and observable by the adversary, then this would give her a considerable advantage to later breach the privacy of communications.

As such, the route proposal mechanism must ensure the following properties against the considered adversary:

Route Proposal Homogeneity: A self-proposal $\texttt{RP}(X{\leftrightarrow}R{\rightarrow}R)$ and a relayed proposal $\texttt{RP}(X{\leftrightarrow}Y{\rightarrow}R)$ towards the same end-receiver R must be indistinguishable, except for the proposer and the end-receiver.

Route Proposal Indistinguishability: A route proposal $\texttt{RP}(X{\leftrightarrow}Y{\rightarrow}R)$ towards end-receiver R must be indistinguishable from a route proposal $\texttt{RP}(X{\leftrightarrow}Y{\rightarrow}R')$ towards a different end-receiver R', except for the end-receiver itself.

Propagation Untraceability: Route proposals $\texttt{RP}(X{\leftrightarrow}Y{\rightarrow}R)$ and $\texttt{RP}(X'{\leftrightarrow}Y'{\rightarrow}R)$ towards the same end-receiver R, appearing at different places and/or times in the network, must be unlinkable.

Return Trip Untraceability: Link messages involved in a given route proposal must be unlinkable, in the sense of MU-tracing.

The first property ensures that proposees receiving a route proposal can not conclude whether the proposer is the end-receiver or not. If that were the case, RA would be directly broken: neighbors of R (who know its IP address) would know which circuits, and which messages, are flowing towards R. The second property is complementary. It ensures that *relayed* route proposals do not leak information on the IP address or real-world identity of the end-receiver it relates to. The third means that it is impossible to follow the propagation of the route proposals related to a particular end-receiver, and thus from inferring the routes built. The last one simply requires messages involved in the round trip from proposer to end-receiver (in the case of a relayed proposal) to be untraceable, in the same idea as MU-tracing for payload messages. This is also to avoid the created routes from being observable.

All four are defined w.r.t. the *bit pattern* of link messages (and the cryptographic material they carry) involved in route proposals. That is, these properties are formulated independently from TAR, in the same way as MU in Section 1.4.1, and for similar reasons (*i.e.* the impossibility to prove these properties in presence of traffic analysis attacks, as further discussed in Chapter 5).

4.5.2. Pseudonyms: Form and Computation

Pseudonyms, in order to be meaningful, must ensure certain properties. They must be *secure*, abstractly meaning that they conceal the real-world identity of the end-receiver

they designate. Also, every time X receives a route proposal towards R, X should obtain the same pseudonym $PS_{X\to R}$. This last point calls for a deterministic computation of $PS_{X\to R}$, from values specific to R and X.

In practice, the pseudonym of node X towards end-receiver R is defined as follows:

$$PS_{X\to R} = \mathsf{h}(dst_R{}^{src_X}) \tag{4.9}$$

Where $\mathsf{h} : \mathbb{G} \to \{0,1\}^n$ is the SHA-3 hash function, the terms $dst_R \in \mathbb{G}$ and $src_X \in \mathbb{Z}_q^*$ are values generated by (and secret to) R and X respectively, and $\mathbb{G}$ is the same group used in the Elgamal scheme. More exactly, for any given node X, src_X and dst_X are long-lived values, that can be regarded as its *identifiers* as end-sender and end-receiver in the network.

This way of computing pseudonyms achieves the following security properties: (i) for $R' \neq R$, $PS_{X\to R'} \neq PS_{X\to R}$ with high probability (preventing X from mistaking an end-receiver for another); (ii) it is not possible for X to recover dst_R from $PS_{X\to R}$ (which would ultimately allow X to impersonate R); and (iii) for two nodes X and X', it is impossible to know that $PS_{X\to R}$ and $PS_{X'\to R}$ actually designate the same node, and thus impossible to reduce relationship pseudonyms to simple pseudonyms. Respectively denoted *uniqueness*, *one-wayness*, and *indistinguishability*, these properties are formally defined and proved in Chapter 5. They mainly rely on the properties of the hash function used, *i.e.* that SHA-3 can be used to produce outputs indistinguishable from a truly random function. Additionally, note that, because this way of computing pseudonyms does not rely on the end-receiver's IP address or real-world identity, it is impossible for nodes to make the link between pseudonyms and real-world identities.

The computation of $PS_{X\to R}$ during a route proposal consists in a three-message secure multi-party computation (SMPC) protocol between X and R. During the process, however, X does not learn dst_R and R does not learn src_X. For that, the homomorphic properties of the Elgamal ciphertext are leveraged. All network considerations left aside, if there exists a direct and secure communication channel between X and R, this is realized by the sub-protocol depicted in Fig. 4.5, using the ScExp operation of the Elgamal scheme.

Intuitively, this short SMPC protocol is *secure* (meaning that src_X and dst_R stay secret to their respective owner) by the IND-CPA property of the Elgamal scheme. Chapter 5 provides a formal proof. Its actual realisation *inside* the network, and in particular, when X and R are not neighbors, is described in the next section.

4.5.3. Route Proposals in Details

This section details the functioning of route proposals, beginning with self-proposals, and then describing relayed ones. It also shows how the SMPC protocol from Fig. 4.5 can be ported into the network, and finally shows how the route proposal mechanism fulfills the properties listed in Section 4.5.1. For simplicity, the details on the decision of accepting and refusing route proposals are only treated in the next section.

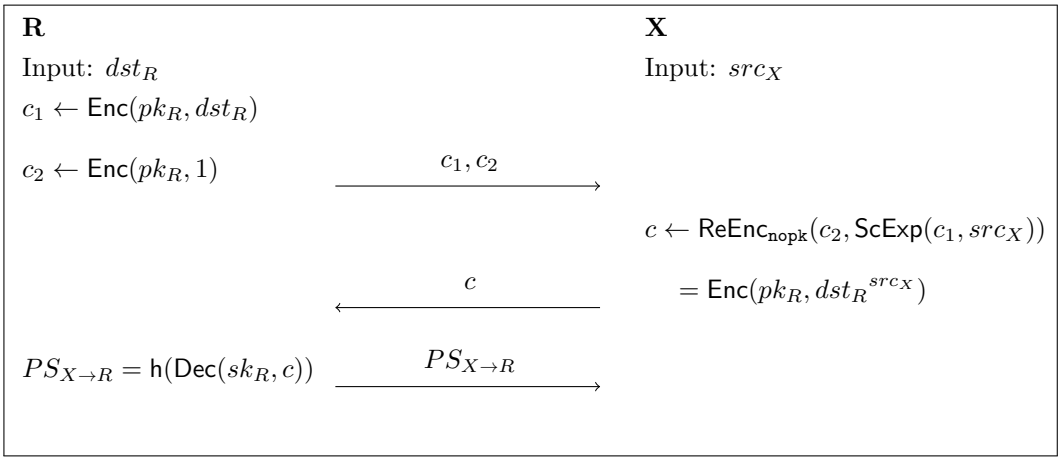

Figure 4.5. – Two-party Computation of $PS_{X \to R}$

4.5.3.a) Self-Proposals

When R self-proposes, it sends a link message to all its neighbors. Each of them answers, since they can not know from this first message whether they need this route or not. Thus, a self-proposal $\mathtt{RP}(X{\leftrightarrow}R{\rightarrow}R)$ takes place between R and each of its neighbors X. Concretely, a route proposal $\mathtt{RP}(X{\leftrightarrow}R{\rightarrow}R)$ for one particular X consists in a three message exchange, depicted in Fig. 4.6. It realises the SMPC from Fig. 4.5, with the following differences: it uses well-formed link messages (with an encrypted flag `rtprop` and a circuit identifier cid), it introduces a key pk_X^{tmp}, the pseudonym $PS_{X \to R}$ is encrypted in the last message, and there are two additional Elgamal ciphertexts $\mathsf{Enc}(pk_R, pk_X^{tmp})$ and $\mathsf{Enc}(pk_X^{tmp}, 1)$. The flag informs the correspondents that this link message relates to a route proposal. The cid is randomly chosen by R among circuit identifiers not already in use between R and X (note that a different cid can be chosen for each neighbor). This term identifies both the newly-formed link, and the ongoing route proposal. The flag and the cid value are encrypted with the keys k_{XR} and k_{RX}, generated from the initial DHKA performed at network setup. Then, because R can not simply send $PS_{X \to R}$ in the clear over the network in the last message, and because R does not know X's public key, a key pk_X^{tmp} is introduced . This is a temporary public key, generated by X specifically for this route proposal. In the second message, X sends it encrypted under pk_R (using $\mathsf{PlainMult}(c_2, pk_X^{tmp})$) . R subsequently answers with an encryption of $PS_{X \to R}$ under pk_X^{tmp}. For reasons that will become clear with the explanation of relayed proposals, R also sends an encryption of one under pk_X^{tmp}.

At the outcome of the route proposal, if X accepts the route, the following entry is created in X's routing table, where the ciphertexts c_1 and c_2 are denoted by $c_{\mathtt{prop}X \to R}$ and

$$
\begin{array}{lll}
m_1 : \{\texttt{rtprop}\|cid\}_{k_{RX}} & \underbrace{\mathsf{Enc}(pk_R, dst_R)}_{c_1} & \underbrace{\mathsf{Enc}(pk_R, 1)}_{c_2} \\
m_2 : \{\texttt{rtprop}\|cid\}_{k_{XR}} & \underbrace{\mathsf{Enc}(pk_R, dst_R{}^{src_X})}_{c} & \mathsf{Enc}(pk_R, pk_X^{tmp}) \\
m_3 : \{\texttt{rtprop}\|cid\}_{k_{RX}} & \mathsf{Enc}(pk_X^{tmp}, \underbrace{PS_{X\to R}}_{PS}) & \mathsf{Enc}(pk_X^{tmp}, 1)
\end{array}
$$

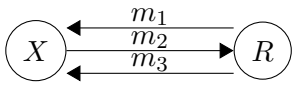

Figure 4.6. – Messages Involved in a Self-Proposal $\texttt{RP}(X{\leftrightarrow}R{\to}R)$

$c_{\texttt{one}X\to R}$, to mark the fact that those are ciphertexts used by X w.r.t. a route towards R.

Prev. hop	PS	$c_{\texttt{one}}$	$c_{\texttt{prop}}$	Next Hop
$\emptyset$	$PS_{X\to R}$	$\underbrace{\mathsf{Enc}(pk_R, 1)}_{c_{\texttt{one}X\to R}}$	$\underbrace{\mathsf{Enc}(pk_R, dst_R)}_{c_{\texttt{prop}X\to R}}$	R, cid

X does not tell R whether it accepts the route or not. Therefore, R always adds (X, cid) as previous hop in its routing entry corresponding to itself. If necessary, R can discard this previous hop if it stays unused for a long period of time.

4.5.3.b) Relayed Proposal

Once X has accepted a proposal, it may relay it (according to the route proposal policy). Assume that X does so. It thus now assumes the role of *proposer*. Node X begins by sending a link message to all its neighbors. Let X' be one of these neighbors. The relayed proposal $\texttt{RP}(X'{\leftrightarrow}X{\to}R)$ is carried out similarly to a self-proposal, the main difference being that the interaction between proposee X' and end-receiver R must be relayed back and forth by X (and, more generally, by every node between proposer and end-receiver). Also, the ciphertexts are now encrypted under a *product* of public keys, that are accumulated or removed during the return trip.

The proposal by X of the route it just learned to one of its neighbors X' consists in the exchange depicted in Fig. 4.7.

The first difference between a relayed proposal and a self-proposal is that X sends a first message with ciphertexts c'_1 and c'_2 for dst_R and 1 encrypted under $pk_{X,R} \coloneqq pk_Y \cdot pk_R$. The node X obtains these by running the $\mathsf{KeyMult}$ operation with sk_X on the ciphertexts $c_{\texttt{prop}X\to R}$ and $c_{\texttt{one}X\to R}$ learned during R's self-proposal. Additionally, X must re-encrypt the ciphertexts so as to realise *propagation untraceability*, *i.e.* so that (c_1, c_2) can not be linked to (c'_1, c'_2) (except by X itself obviously). To summarise:

$$
\begin{aligned}
c'_1 &\leftarrow \mathsf{KeyMult}(sk_X, \mathsf{ReEnc}_{\mathsf{nopk}}(c_{\texttt{one}X\to R}, c_{\texttt{prop}X\to R})) \\
c'_2 &\leftarrow \mathsf{KeyMult}(sk_X, \mathsf{ReEnc}_{\mathsf{one}}(c_{\texttt{one}X\to R}))
\end{aligned}
$$

$$
\begin{array}{lll}
m_1 : \{\texttt{rtprop}\|cid'\}_{k_{XX'}} & \underbrace{\mathsf{Enc}(pk_{X,R}, dst_R)}_{c'_1} & \underbrace{\mathsf{Enc}(pk_{X,R}, 1)}_{c'_2} \\
m_2 : \{\texttt{rtprop}\|cid'\}_{k_{X'X}} & \underbrace{\mathsf{Enc}(pk_{X,R}, {dst_R}^{src_{X'}})}_{c'} & \mathsf{Enc}(pk_{X,R}, pk^{tmp}_{X'}) \\
m_3 : \{\texttt{rtproprelay}\|cid\|rcid\}_{k_{XR}} & \mathsf{Enc}(pk_R, {dst_R}^{src_{X'}}) & \mathsf{Enc}(pk_R, pk^{tmp}_{X,X'}) \\
m_4 : \{\texttt{rtproprelay}\|cid\|rcid\}_{k_{RX}} & \mathsf{Enc}(pk^{tmp}_{X,X'}, \underbrace{PS_{X'\to R}}_{PS}) & \mathsf{Enc}(pk^{tmp}_{X,X'}, 1) \\
m_5 : \{\texttt{rtprop}\|cid'\}_{k_{XX'}} & \mathsf{Enc}(pk^{tmp}_{X'}, \underbrace{PS_{X'\to R}}_{PS}) & \mathsf{Enc}(pk^{tmp}_{X'}, 1)
\end{array}
$$

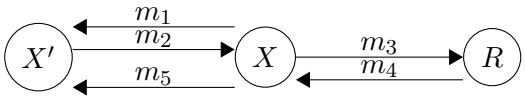

Figure 4.7. – Messages Involved in a Relayed Proposal $\texttt{RP}(X' \leftrightarrow X \rightarrow R)$

The main characteristic of a relayed proposal is the *return trip* from X to R. In the example, this return trip only takes two link messages (the third and fourth one in Fig. 4.7), but in general, it takes $2l$ link messages, where l is the number of hops (in the topology graph) between the proposer and the end-receiver. The implementation of the return trip while ensuring privacy poses two main challenges, that the exchange in Fig. 4.7 solves.

(i) The need to use circuits in a *reverse way* (with the fourth message).
Although using circuits in this fashion is trivial in *e.g.* Tor, where nodes have a one-to-one mapping from previous to next hops, in the present protocol, a node may have several previous hops. And in particular, when X makes a proposal, it must concurrently handle one proposal towards R for each of its neighbors. When X receives the fourth message from R, it thus needs a *reverse circuit identifier* $rcid$ value to be able to know that the message relates to an ongoing proposal with X' specifically. This value is generated by X and repeated by R on the way back. In the general case, when there are l hops between proposer and end-receiver, each of the l relay nodes will independently generate its own $rcid$ value.

(ii) The need to avoid the tracing of ciphertexts in the return trip (and in particular, of the encrypted $PS_{X\to R}$ on the way back from R to X') so that *return trip untraceability* holds.
For this, we again use URE and the technique of encryption under a product of public keys. Note that on the way *forward* (from X' to R), ciphertexts are encrypted under a product of public keys (here, $pk_X \cdot pk_R$) so that an adversary controlling R and thus knowing sk_R can not trace them; while on the way *back*, it is to prevent tracing by X' that they are encrypted under a product of public keys (here, $pk^{tmp}_X \cdot pk^{tmp}_{X'}$).

In more detail, the cryptographic operations performed by X (and all relay nodes on the return trip in the general case) are as follows. On the way *forward*, given $c_1 \coloneqq \mathsf{Enc}(pk_{X,R}, {dst_R}^{src_{X'}})$ and $c_2 \coloneqq \mathsf{Enc}(pk_{X,R}, pk^{tmp}_{X'})$,

(1) Generate $(pk^{tmp}_X, sk^{tmp}_X) \leftarrow \mathsf{KeyGen}(1^\lambda)$

(2) Compute $c'_1 \leftarrow \mathsf{ReEnc}_{\mathtt{nopk}}(c_{\mathtt{one}X\to R}, \mathsf{Dec}(sk_X, c_1))$

(3) Compute $c'_2 \leftarrow \mathsf{ReEnc}_{\mathtt{nopk}}(c_{\mathtt{one}X\to R}, \mathsf{Dec}(sk_X, \mathsf{PlainMult}(c_2, pk^{tmp}_X)))$

Send the resulting ciphertext to R (as depicted in line 3 of Fig. 4.7). On the way back, given $c_1 \coloneqq \mathsf{Enc}(pk^{tmp}_{X,X'}, PS_{X'\to R})$ and $c_2 \coloneqq \mathsf{Enc}(pk^{tmp}_{X,X'}, 1)$,

(1) Compute $c'_2 \leftarrow \mathsf{ReEnc}_{\mathtt{one}}(\mathsf{Dec}(sk^{tmp}_X, c_2))$

(2) Compute $c'_1 \leftarrow \mathsf{ReEnc}_{\mathtt{nopk}}(c_2, \mathsf{Dec}(sk^{tmp}_X, c_1))$

Send the resulting ciphertext to X' (as depicted in line 4 of Fig. 4.7). Notice the use of *temporary* public keys: they accumulate in a ciphertext on the way forward with a $\mathsf{PlainMult}$ operation; R then uses $pk^{tmp}_X \cdot pk^{tmp}_{X'}$ to encrypt $PS_{X'\to R}$; and on the way back, relay nodes run Dec with their temporary secret key, ensuring that X' gets the ciphertext back encrypted solely under $pk^{tmp}_{X'}$.

Ultimately, if X' accepts the route, X adds (X', cid') as previous hop in its entry towards R, and X' adds the following entry to its routing table:

Prev. hop	PS	$c_{\mathtt{one}}$	$c_{\mathtt{prop}}$	Next Hop
$\emptyset$	$PS_{X'\to R}$	$\underbrace{\mathsf{Enc}(pk_{X,R}, 1)}_{c_{\mathtt{one}X'\to R}}$	$\underbrace{\mathsf{Enc}(pk_{X,R}, dst_R)}_{c_{\mathtt{prop}X'\to R}}$	X, cid

4.5.3.c) Security of Route Proposals Against Traffic Analysis

In the described route proposal mechanism, *route proposal homogeneity*, *route proposal indistinguishability*, *propagation untraceability*, and *return trip untraceability* are fulfilled. Propagation and return trip untraceability have already been discussed extensively. Route proposal homogeneity is ensured since, from the point of view of the proposer, the exact same cryptographic material and link messages are sent and received. Route proposal indistinguishability is also ensured, because by the security properties of the Elgamal scheme, the proposer and proposee get no information on R. In particular, note that, when X relays the proposal in Fig. 4.7, although it knows that R is the next hop, it can not be sure that it is the end-receiver. Lastly all four properties hold against a global network observer, intuitively because, all an observer sees are link messages carrying random-looking data.

All these claims are formally proven in Chapter 5. That is, it is shown that these four properties are ensured *cryptographically*, in accordance with their definition in Section 4.5.1.b. However, without any additional protection, they clearly do not hold under traffic analysis attacks. In particular, a self-proposal can be distinguished from a relayed one since it has no return trip and thus takes much less time. Here, we briefly

give additional measures to thwart traffic analysis attacks aimed at downgrading the security of route proposals. These measures are however *not* included in formal proofs for reasons exposed in the next chapter.

To prevent traffic analysis attacks on the return trips, we actually include routing messages in the message re-ordering mechanism presented in Section 4.4.2. Meaning that all messages that X wants to send as part of a route proposals are not simply sent to neighbors, but placed in its pools along with dummy and payload messages. As a result, payload and routing messages are mixed together (this is possible since, by design, they are of the exact same form). This makes actual communications indistinguishable from route proposals for network observers, and also has the advantage of providing nodes with further cover traffic. On the downside, this means that the completion of topology dissemination (*i.e.* making each node learn about each other) can take very long. This is however the price to pay: intuitively, the topology dissemination has to be *slow* in order for it to be stealthy and avoid being subject to traffic analysis attacks.

Applying message-reordering on route proposal messages contributes directly to protecting return trip untraceability, but it also helps thwart basic timing analysis attacks on route proposal homogeneity, since it introduces latency in the return trip. Analogously, this helps regarding route proposal indistinguishability, in the sense that it does not let the adversary easily distinguish two relayed route proposals based on her estimated distance to different end-receivers (in terms of hops). Lastly, it helps regarding propagation untraceability. Indeed, when a node decides to relay a proposal, it places the adequate link messages in its pools, and the batching strategy delays the actual relaying for some time (and a different time for each neighbor).

Lastly, there is a particular threat to route proposal homogeneity: if a node X joins a network where topology dissemination has already been completed, it will begin by self-proposing, and be trivially detected by its neighbors. Since these neighbors see the IP address of X, and know that it is the end-receiver, they ultimately break RA. This phenomenon also appears in the Crowds protocol [RR98], and can be solved in a similar manner, by having all or a subset of nodes around X also begin a new cycle of route proposals under a new identity (with a new *dst* value). This allows the newcomer to blend in this subset.

4.5.4. The Route Proposal Policy: Accepting or Refusing the Routes

In traditional routing protocols, nodes often learn several possible routes to every other node, but select only one or a few of them, *e.g.* based on performance metrics such as the length of the route, or the bandwidth it offers. Nodes also are careful not to create routing loops, in which a message could get stuck, indefinitely going in circle through the same set of nodes. In the present protocol, this is the role of the route proposal policy.

This work does not propose a concrete policy. But the present section discusses how information and metrics on the routes may be communicated to the proposee, and how based on these information, it can decide to accept or refuse a proposal, and relay it or not.

Note that we do not undermine the importance of the route proposal policy, which is

essential to ensure that topology dissemination does not stop before every node learn about every other. It also plays an important role in the security of the protocol. First because it must avoid revealing too much information on circuits so as not to breach privacy properties. Secondly, it must not be *simplistic*: in particular, deterministic policies must be avoided, because given a topology graph, they will always yield the same circuits. Lastly, because in terms of privacy, some routes are preferable to others: shortest paths may make the network more efficient, but are easy to infer for anyone aware of (portions of) the topology graph.

4.5.4.a) Decision Process

The described protocol already provides nodes with one basic piece of information: by the properties of pseudonyms, nodes can count the routes they have towards a given (anonymous) end-receiver. The policy can dictate that a node should not have more than *e.g.* three routes towards the same end-receiver. A proposee would thus refuse a route proposal towards a pseudonym for which it has already 5 routing table entries. However, recall that proposees only get $PS_{X \to R}$ at the very end of the route proposal. It can thus make its choice only *a posteriori*. After the accept/refuse decision, the policy should also say whether the node should relay the proposal. This can be based on a simple coin flip, or on the number of recently received proposal for a given end-receiver for instance.

To communicate other information about the route, the sub-protocol corresponding to route proposal must be extended: additional messages must be sent, to carry information on the circuits. To minimize the information leak on routes, we employ the following measures: this information is concealed in Elgamal ciphertexts, homomorphic operations are used to process it, and eventually, a proposee only obtains a *yes/no* answer on whether it should accept the route, and another on whether it should relay it or not. This approach makes it possible to let the proposee choose its own route-selection criteria, and yet reveals only two bits of information. In practice, this approach is realised by designing a specific SMPC protocol between proposee and end-receiver for each type of information to be communicated on circuits.

4.5.4.b) Privately Communicating Information on Routes

Before giving a generic methodology for taking into account any metric or information about the route, we give a concrete example of SMPC focused on the hop count metric.

The most basic metric characterising a route is its length l, and routing protocols must limit it to some number l_{max} of hops. We show how to extend route proposals with additional messages in order to allow proposees to obtain a boolean indicating if the route is longer than l_{max} or not. This corresponds to a private *range test* protocol, for which SMPC constructions already exist [Pen+06]. Hereby, we propose a basic construction that integrates well with route proposals. First of all, when R self-proposes, it communicates a ciphertext $c_\ell = \mathsf{Enc}(pk_R, g^1)$ (since at that time, the route is only one-hop long). This ciphertext is meant to be treated similarly to $c_{\texttt{one}}$ and $c_{\texttt{prop}}$, *i.e.* encrypted under the product of all relay nodes' keys, re-encrypted, and included in all relayed versions of

the proposal. Additionally, a node that relays a proposal performs $c'_\ell = \mathsf{PlainMult}(c_\ell, g) = \mathsf{Enc}(g^{l+1})$ in order to update the route length. In the general case, a proposee X receiving a proposal from Y towards R receives $c_\ell = \mathsf{Enc}(pk_{Y,\dots,R}, l)$, which it uses in the following way to know if $l > l_{max}$, for a value of l_{max} publicly fixed. X begins by computing $\mathsf{CtxtMult}(\mathsf{Enc}(pk_{Y,\dots,R}, l_{max}), c_\ell^{-1}) = \mathsf{Enc}(pk_{Y,\dots,R}, g^{l_{max}-l})$. This latter ciphertext is sent back to R along with the ciphertext for ${dst_R}^{src_X}$. R then receives the ciphertext, can decrypt it, and gets $g^{l_{max}-l}$. It answers with $c_b \leftarrow \mathsf{Enc}(pk^{tmp}, g^b)$, where $b = 1$ if $g^{l_{max}-l} \in \left[g^0, g^{l_{max}-1}\right]$, and $b = 0$ otherwise. The ciphertext c_b travel back to X similarly to the one encrypting $PS_{X\to R}$. X can decrypt it, and accepts the route only if $g^b = g$.

This small protocol reveals only one bit of information to X, but possibly leaks to R the exact distance of the proposee. This is deemed acceptable, because circuits do not correspond to a unique end-sender: payload message received by R incoming on a particular circuit may have been end-sent by any of the nodes in the circuit.

More generally, any metric or information can be communicated to the proposee following the same ideas as in the hop count example: an initial value is encrypted and included in self-proposals, the information is then (homomorphically) accumulated as route proposals propagate, proposees homomorphically process ciphertexts, and obtain a simple *yes/no* answer by collaborating with the end-receiver R (the only party able to fully decrypt and get the piece of information). There is often the choice between using a very efficient SMPC protocol that leaks information to R (but not to X), and using a more complex one that ensures that both X and R learn only one bit of information. For better privacy, the latter approach should be favored.

The more complex the route proposal policy is, the more it costs in terms of number of transmitted messages on the return trip between proposee and end-receiver. Indeed, each piece of information must be included in a different ciphertext in the initial message of route proposals. However, note that X can sometimes reduce all these ciphertext to one unique ciphertext encrypting g^b before sending it to R, by homomorphically applying an adequate boolean formula. If not, this can be done by R, saving at least bandwidth for the way back of the return trip.

4.5.4.c) Routing Loops

An important component of a routing protocol is the prevention of routing loops in the forwarding process. However, to the best of our knowledge, there is no existing privacy-preserving solution to test the presence of loops applicable to the present protocol. Indeed, the data structure to store information on nodes that are already on the route must: (i) be of constant size (to avoid leaking the length of the route), (ii) fit into one or a few Elgamal ciphertexts, and (iii) be manipulable through homomorphic operations (so as to insert an element in the structure, and test membership). Previous works [Don+09; Boc+12] proposed the use of a bloom filter. However, this structure can not be manipulated by homomorphic operation unless its bits are encrypted separately. The same goes with the use of polynomials to represent set operations [KS05]: their coefficient must be separately

encrypted, yielding too many ciphertexts to handle in a route proposal. More generally, and to our knowledge, no such (efficient) data structure exists and no existing SMPC protocol can appropriately prevent the formation of routing loop.

However, this is actually not an issue in the present protocol. Indeed, by the way routing tables are constructed and used, messages can not be stuck indefinitely in a loop. That is, a circuit may indeed go twice through a same node, but even so, messages do not get stuck in an infinite loop, thanks to the unidirectional use of circuits. We choose to tolerate such loops. Although they clearly impacts efficiency, they also brings more privacy, at least compared to a route proposal policy that builds shortest routes.

4.6. Oriented Communications: Alice Contacts Bob

This section presents the final building block of the protocol, that enables what is called *oriented communication*. It can be seen as an extension of the protocol, since the latter can fully function without this final block. Indeed, given what has already been presented, nodes can communicate with anonymous end-receivers that they know under their pseudonyms. This is sufficient for applications in which individuals simply look for *a* communication partner, but not for one in particular (in an online game, or a file-sharing application for instance). It is also sufficient for a use of the network in a Tor fashion with some nodes acting as *exit nodes*. The latter application however implies a client-server architecture in which RA can not be ensured (and also necessitates *exit policies* preventing misuse of the anonymity provided by the network, which is an issue in itself).

Still, in view of the informant-journalist scenario, the functionality provided by the protocol is not sufficient: an informant Alice must be able to open a bi-directional communication channel with a *specific* journalist Bob of its choice (while remaining anonymous to Bob). This mode of communication is hereby called *oriented*. As they are, routing tables do not contain information that could help Alice in this endeavor. This section fills that gap. It also analyses the SA, RA, SU, MU and TAR properties w.r.t. oriented communication sessions, showing that the informant and journalist obtain the desired anonymity.

4.6.1. Intuition

What oriented communications must essentially achieve is to translate "Bob" into Bob's pseudonym(s) in the network. The simplest solution would be for Bob to publish its (possibly certified) anonymous receiver identity, dst_B. With this, Alice can compute $PS_{A\to B} = \mathsf{h}({dst_B}^{src_A})$ and contact Bob. This however consists in a breach of RA: every node X, when sending or relaying a message towards $PS_{X\to B}$, will know that the message is directed to Bob. A slightly better solution would be to communicate dst_B only to Alice. In this case, only Alice breaks RA towards Bob. Still, we reject this solution for the infringement to privacy it implies. Instead, we propose a solution based on the use of an *indirection node* I, a regular network node (not necessarily trusted), making the junction between Alice and Bob. This approach, somewhat inspired from *rendez-vous* points in

past works (including Tor) [DMS04; Nez+09], tilts the privacy/efficiency trade-off in favor of privacy.

The solution relies on a *secret sharing* scheme applied to Bob's dst_B value. In the class of secret sharing schemes of interest here [Sha79], a secret value v is split into two *shares*. Given one share, nothing can be learned about the secret, but with two shares, that value can be recovered. Here, Bob gives one share of dst_B to I, and the other to Alice. Through yet another SMPC protocol, Alice and I compute $PS_{I\to B}$, allowing I to find a route towards Bob. Alice then routes each payload message meant for Bob towards I, which then forwards it on one of its routes towards $PS_{I\to Bob}$. As a result, Alice knows the real-world identity of Bob (but not $PS_{Alice\to B}$), and I knows $PS_{I\to B}$ (but not Bob's real-world identity). That is, the knowledge is divided, and no one party can make the connection between real-world and anonymous network identities. The solution additionally requires a setup step, during which Alice and Bob exchange some information. That is, Bob must communicate the shares to Alice *in some way*, one of which is encrypted so that only I can access it.

An implication of the proposed solution is that two *levels* of routing, and two kinds of routes appear: the *simple routes*, built during topology dissemination, and the full oriented communication routes, consisting of two simple routes, respectively denoted the first and second *leg*. This also means that there are two levels of SA and RA. So far, the described protocol provides SA and RA for end-senders and end-receivers of *simple routes*. However, the ultimate goal of the protocol is to provide SA and RA for end communicants of oriented communication sessions. We will see that privacy of simple routes realises privacy of the oriented communication routes.

4.6.2. Detailed Description

Hereby, Alice and Bob are assumed to respectively run node A and B in the network. Bob constructs the two shares of dst_B by sampling the first share $sh_1 \leftarrow\!\!\$\ \mathbb{G}$, and setting the second share to $sh_2 = dst_B/sh_1 \in \mathbb{G}$. One share reveals nothing on dst_B, since $\mathbb{G}$ is cyclic, and for $i = 1$ or 2, $\{e \cdot sh_i \mid e \in \mathbb{G}\} = \mathbb{G}$, meaning that, given a share sh_i, dst_B could still be any element of $\mathbb{G}$.

The choice of I must be made by Alice. It can not be made by Bob, since Bob would consequently be unable to communicate the identity of I to Alice, because of the *indistinguishability* of pseudonyms. Alice chooses I by selecting a random entry $E_{A\to I} = \langle PrevHops, PS_{A\to I}, c_{\texttt{one}A\to I}, c_{\texttt{prop}A\to I}, NextHop\rangle$ in her routing table. Note that Alice choses the indirection node, but actually does not know its identity or IP address.

Alice and Bob then make contact *in some way*: Alice gives $c_{\texttt{one}A\to I}$ to Bob, who answers with sh_1 and $\mathsf{Enc}_{\mathsf{nopk}}(c_{\texttt{one}A\to I}, sh_2) = \mathsf{Enc}(pk_{Z_1,Z_2,\ldots,I}, sh_2)$ (where $Z_1, Z_2, \ldots$ are the relay nodes between Alice and node I). Additionally, Alice must communicate an oriented communication identifier *ocomid*, and shared key k to Bob (the latter will be used to conceal the payload data from the node I). This preliminary exchange can be performed *outside of network*, as proposed in the existing protocols such as MIAB and Pung [IKV13; AS16]. But it can also be performed by *exceptionally* using the anonymous network in a client-server fashion, for Alice to contact some (web) server publicly known to be run by

Bob. The advantage of the latter option is that the anonymous network ensures Alice's privacy even for this preliminary exchange, but reveals that *someone* wants to contact Bob.

Regardless of the method used for the interaction between Alice and Bob, once Alice has sh_1 and $\mathsf{Enc}(pk_{\ldots,I}, sh_2)$, she can engage in a SMPC protocol with I, denoted the *oriented communication initialisation*. The goal in this interaction is for I to obtain $PS_{I\to B}$. In the process, neither Alice nor I learns dst_B as long as they do not collude. The SMPC is run *inside* the network. In particular, Alice and I communicate using the circuits built during topology dissemination. That is, Alice sends end-to-end `payload` messages in order to contact I. However, for I to answer, simple `payload` messages are not sufficient: I does not know route towards Alice, nor her pseudonym $PS_{I\to A}$. Therefore we use a construction similar to the *return trip* in route proposals, with *reverse circuit identifiers rcid*. That is, I can answer through the *reverse route* that Alice uses.

We in fact decide to re-use the `rtproprelay` routing messages in exactly the same way as in route proposals. This makes the oriented communication initialisation *look like* a return trip part of a route proposal, and makes oriented communications harder to detect. Figure 4.8 depicts how this is achieved. The first part presents the sequence of end-to-end messages that realise the oriented communication initialisation (where I obtains $PS_{I\to B}$), and the second part is the actual sending of payload data m, flowing from Alice to I and then from I to Bob. For conciseness and clarity, many details are omitted. Although Alice and I are not necessarily neighbors, the role of relay nodes between them is not represented; cryptographic operations carried out by Alice and I are referenced by markers and described below the figure; and messages are given with a generic end-to-end message notation $\langle \texttt{type}, \mathsf{Enc}(data_1), \mathsf{Enc}(data_2)\rangle$, where $\texttt{type} \in \{\texttt{payload}, \texttt{rtproprelay}\}$. The ciphertexts in these messages implicitly undergo the same processing as described earlier in this chapter (see Section 4.3.2 for `payload` messages, and Section 4.5.3.b for `rtproprelay` ones).

The initialisation of an oriented communication thus requires nine end-to-end messages in the network, and the sending of each payload m_i requires two end-to-end messages each. Notice how, in each end-to-end messages, the two Elgamal ciphertexts $\mathsf{Enc}(data_1)$, $\mathsf{Enc}(data_2)$ are put to use. All messages sent by Alice consist of a first ciphertext containing control data (the *ocomid* and a *counter*), allowing I to link and re-order messages (indeed, messages are expected to arrive out of order, by the message re-ordering mechanism); and only the second ciphertext carries data directly useful to the computation of $PS_{I\to B}$. Likewise, because I can only answer Alice using the *reverse route*, it can send only one *useful* ciphertext at a time: it *must* answer with `rtproprelay` messages, and such messages must contain an encryption of one in the second ciphertext.

The operations carried out by nodes at markers (1), (2), (3), and (4) in Fig. 4.8 are as follows, knowing that Alice starts with sh_1, $c_{sh_2} = \mathsf{Enc}(pk_{Z_1,Z_2,\ldots,I}, sh_2)$:

(1) Alice generates $(pk_A^{ocom}, sk_A^{ocom}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $(pk_A^{tmp}, sk_A^{tmp}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. She encrypts all plaintexts, including pk_A^{ocom} and pk_A^{tmp}, using the $\mathsf{Enc}_{\texttt{nopk}}(c_{\texttt{one}A\to I}, \cdot)$ operation, as any plaintext meant to be sent towards I. Note that c_{sh_2} already encrypts sh_2 under the adequate (product of) public key(s), and can be sent as is

(1)	m_0 :	`payload`,	$\mathsf{Enc}(\texttt{ocom}\|\|ocomid\|\|0)$,	$\mathsf{Enc}(pk_A^{ocom})$
	m_1 :	`payload`,	$\mathsf{Enc}(\texttt{ocom}\|\|ocomid\|\|1)$,	c_{sh_2}
	m_2 :	`payload`,	$\mathsf{Enc}(\texttt{ocom}\|\|ocomid\|\|2)$,	$\mathsf{Enc}(c_{sh_1}[0])$
	m_3 :	`payload`,	$\mathsf{Enc}(\texttt{ocom}\|\|ocomid\|\|3)$,	$\mathsf{Enc}(c_{sh_1}[1])$
	m_4 :	`rtproprelay`$(rcid)$,	$\mathsf{Enc}(\texttt{ocom}\|\|ocomid\|\|4)$,	$\mathsf{Enc}(pk_A^{tmp})$
	m_5 :	`rtproprelay`$(rcid')$,	$\mathsf{Enc}(\texttt{ocom}\|\|ocomid\|\|5)$,	$\mathsf{Enc}(pk_A^{tmp})$
(2)	m_6 :	`rtproprelay`$(rcid)$,	$\mathsf{Enc}(c[0])$,	$\mathsf{Enc}(1)$
	m_7 :	`rtproprelay`$(rcid')$,	$\mathsf{Enc}(c[1])$,	$\mathsf{Enc}(1)$
(3)	m_8 :	`payload`,	$\mathsf{Enc}(\texttt{ocom}\|\|ocomid\|\|8)$,	$\mathsf{Enc}(PS_{I\to B})$

Initialisation

Payload sending

(4)	m_{8+i} :	`payload`,	$\mathsf{Enc}(\texttt{ocom}\|\|ocomid\|\|8+i)$,	$\mathsf{Enc}(\{m_i\}_k)$
	m'_{8+i} :	`payload`,	$\mathsf{Enc}(\texttt{ocom}\|\|ocomid\|\|8+i)$,	$\mathsf{Enc}(\{m_i\}_k)$

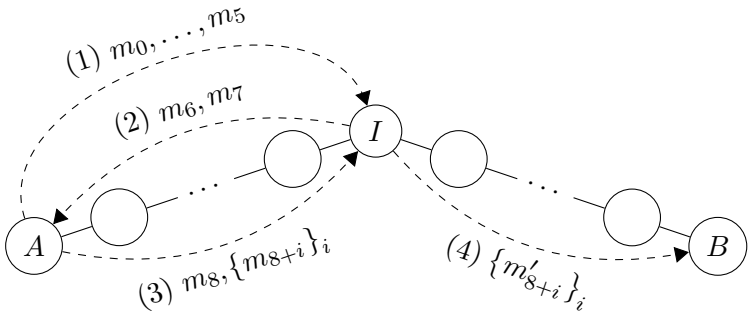

Figure 4.8. – Messages Involved in an Oriented Communication Initialisation

(after a re-encryption). Then, Alice *doubly encrypt* the other share sh_1: once under pk_A^{ocom} (to prevent I from learning it), and once again to be sent in the circuit. That is, Alice computes $c_{sh_1} \leftarrow \mathsf{Enc}(pk_A^{ocom}, sh_1)$. An Elgamal ciphertext being made of two group elements, but only being able to encrypt one at a time, Alice then separately encrypts the first and second components of c_{sh_1} (noted $c_{sh_1}[0]$and $c_{sh_1}[1]$) using $\mathsf{Enc}_{\mathtt{nopk}}(c_{\mathtt{one}A\to I}, \cdot)$. All six messages are then sent *on the same circuit* towards I.

(2) Having received pk_A^{ocom}, $pk_{A,Z_1,Z_2,\ldots}^{tmp}$, sh_2, and c_{sh_1}, I computes:

$$(c[0], c[1]) \coloneqq c \leftarrow \mathsf{ReEnc}_{\mathtt{pk}}(pk_A^{ocom}, \mathsf{PlainMult}(\mathsf{ScExp}(c_{sh_1}, src_I), sh_2^{src_I}))$$

The node I sends back $c[0]$ and $c[1]$ encrypted under $pk_{A,Z_1,Z_2,\ldots}^{tmp}$ in two distinct `rtproprelay` messages along the reverse route, as though those messages were part

of a route proposal return trip. This is why Alice must send two `rtproprelay` messages in the first place: to give the opportunity to I to send back two `rtproprelay` messages, each containing a piece of c.

(3) Alice receives and decrypts $c = (c[0], c[1]) = \mathsf{Enc}(pk_A^{ocom}, {dst_B}^{src_I})$, hashes the result to get $PS_{I \to B}$, and sends it back to I in a regular `payload` message, still using *the same circuit* as before.

(4) Once I knows $PS_{I \to B}$, Alice starts sending payload data, under the form of several $|q|$-bit chunks m_i that fit into Elgamal ciphertexts. These are encrypted first with k, the key shared between Alice and Bob (to conceal the payload from I), and then with $\mathsf{Enc}_{\mathtt{nopk}}(c_{\mathrm{one}A \to I}, \cdot)$, and sent to I again with the same circuit.
When I receives a message with a counter greater to 8, it knows it is payload data. It selects an entry towards $PS_{I \to B}$ in its routing table, and simply relays payload messages to B (all messages are sent on the same circuit to B).

The solution described here only allows to build a unidirectional route from Alice to Bob. To answer, Bob can however use *reverse route* on the whole route (from B back to I back to A). We suggest, however, that this reverse route only be used for Bob to obtain shares of Alice's dst_A value, so that Bob can then make a separate oriented communication initialisation. This ensures a clear separation between Alice's messages, and Bob's answers to them.

4.6.3. Analysis

The correctness of the SMPC is straightforward, since $sh_1^{dst_I} \cdot sh_2^{src_I} = {dst_B}^{src_I}$. Cryptographically speaking, security holds because dst_B is a generator of $\mathbb{G}$ (as any element of $\mathbb{G} \setminus \{1\}$), and assuming that Alice and I do not collude. More specifically, by the IND-CPA property of the Elgamal scheme, the security of the secret sharing, the DL problem, I does not learn anything except $PS_{I \to B}$, and Alice can not recover dst_B nor src_I. The formal security proof can be found in Chapter 5.

The oriented communication mechanism as a whole leaks almost no information to the concerned parties (Alice, Bob, and I), nor to observers and relay nodes. In particular, Alice stays anonymous, even to Bob (though Bob is not anonymous to Alice, of course). I only learns that *someone* wants to communicate with $PS_{I \to B}$. Relay nodes can not know whether the `rtproprelay` messages correspond to a route proposal or to an oriented communication initialisation (note that two different *rcid* values are used for the two `rtproprelay` messages). Relay nodes *do* detect communications by the presence of `payload` messages, but they can not however know whether those are payload messages on the first *leg* (between Alice and I) or on the *second* one (between I and Bob). This makes it harder for corrupted nodes to infer their own location on the route.

As mentioned previously, the oriented communications consist in another *level* of routing. It is on this level, and on oriented communication routes, that the SA, RA, SU, MU, and TAR must ultimately be ensured. The construction of oriented communication

achieves (almost) all of them. SU holds simply because two sessions between the same Alice and Bob do not share any common data. Even the indirection node I differs from session to session. SA (for Alice) and RA (for Bob) hold, because these properties already hold on each leg of the full oriented communication route, and because the oriented communication initialisation conceals the identity of Bob to I. Similarly, TAR and MU hold on each leg. However, they do not completely hold on the full route, because of the position of the indirection node. Indeed, I clearly breaks MU-session, since it can link together all messages using the *ocomid*. Likewise, MU-tracing can be broken by a corrupted Alice and/or Bob colluding with a corrupted indirection node, since the encrypted payloads $\{m\}_k$ are seen by Alice, Bob and I, and are not changed of appearance on the route. The TAR property is also impacted, since I is able to re-order messages. These infringements to the MU and TAR properties are however deemed acceptable, for they do not seem to lead to a breach of SA or RA. In particular, an adversary controlling Bob and I and breaking MU-tracing does not learn much information: for all she know, any node in the network could still be the end-sender, since any node in the network can reach I.

In addition to I fully breaking MU-session, note that all exchanged messages between Alice and I go through the same circuit and the same relay nodes (and likewise for the second leg, between I and Bob). Although this design choice further degrades MU-session, the formal analysis in Chapter 5 shows that it yields better anonymity overall. If necessary, MU-session can be emulated by Alice by initialising several communication sessions with Bob, with different indirection nodes, and to *split* the *flows* of data she needs to send over several channels (as suggested by Serjantov and Murdoch [SM05]).

Finally, note that oriented communications can take a substantial amount of time to be carried out, because all the messages during the oriented communication initialisation are delayed, as any other message, by the message re-ordering mechanism.

4.7. Summary and Discussion

In this chapter, a new Internet overlay protocol for strongly private communications was presented. Combining several existing mechanisms, and introducing new ones, the chapter details how SA, RA, MU, SU, and TAR are achieved. This last section further explains how each protocol component participates in realising these properties. Before concluding, it also presents various interesting properties of the protocol.

Privacy

Table 4.1 summarises all the privacy-enhancing mechanisms or properties of the protocol, and specifies which of SA, RA, MU-session, MU-tracing, or TAR they participate in achieving. If the mechanism is protecting only against external adversaries (and not internal ones), a partial tick "/" is used. The SU property is not included in the table, since it is simply ensured by the absence of common elements across sessions. Also, each mechanism listed in the table is accompanied with a reference to its corresponding section number in the thesis, where the reader may find the relevant details.

		SA	RA	MU		TAR
				sess.	trac.	
Homogeneous Architecture (4.1)		X	X			/
Dummy Messages (4.4.1)		/	/			/
Traffic Rates (4.4.1)		X	X			X
Pseudonyms (4.5.2)			X			
Shared Circuits (4.2.2.b)		X		X		X
Re-encryption (4.3.2)					X	X
Message re-ordering (4.4.2)						X
Split flows (4.6.3)				X		X
Route Prop. (4.5.1.b, 4.5.3.c) (4.5.3.c)	Homogeneity		X			
	Indistinguishability		X			
	Propagation Untrac.		X		X	
	Return Trip Untrac.		X		X	
	Batching w/ payload		X			X

Table 4.1. – Which Mechanism Ensures Which Privacy Property?

Abstractly, the protocol conceals almost everything from all network entities. External adversaries only see random-looking data, from the topology dissemination to the sending of payload data. The knowledge of internal adversaries (corrupted nodes) is limited to the IP address of their neighbors, and the previous and next hop of each circuit they are part of. Additionally, by the relay homogeneity and the use of dummy messages and controlled traffic rates, a corrupted node can not know for sure if the previous hop is the sender or not, and if the next hop is the receiver or not. Pseudonyms, with the procedure initialising oriented communication, are the key to ensure RA, along with the security properties of route proposals. Oriented communications are designed to allow Alice to stay completely anonymous to Bob. MU-tracing is mainly ensured by re-encryption at each hop. MU-session holds first by the shared circuits: a relay can never know if two messages in the same circuit come from the same sender. The possibility to split flows of data over several oriented communication sessions also participates in ensuring MU-session. The TAR property, being very broad, is realised by a collection of mechanism. Roughly, all that participates in MU is also useful for TAR. Dummy messages, and the homogeneous architecture also thwart attacks from external adversaries, and traffic rates attacks from internal ones. But the main key to TAR is of course the message re-ordering mechanism adapted from mixnets. Finally, the whole design of route proposals is eventually aimed at concealing the end-receiver of circuits, and preventing the adversary from inferring the circuits created (and thus ultimately protecting MU-tracing). And the fact that routing and payload messages are batched and re-ordered together can only improve resistance to traffic analysis.

Table 4.1 however presents security mechanisms and their role in an informal manner. As discussed in the next chapter, not all the elements in that table appear in formal proofs (and in particular, TAR can not be proven).

Compared to previous works, note that cascade mixnets often claim relationship anonymity is ensured as long as there is only one non-corrupted node on the path between sender and receiver. In the present protocol, SA, RA, and thus relationship anonymity can hold even if *all* relay nodes are corrupted. Indeed, in the general case, a collusion of corrupted nodes can not even know that it occupies the full path. This is also a property of Crowds and Tarzan, and of homogeneous networks in general.

Conclusion and Insight

In this chapter, we proposed a new Internet overlay for strongly private communications. Building upon existing protocols, Tarzan and mixnets in particular, it ensures *unobservable communications* in a fully distributed network. For that, we adapted the message reordering mechanism from mixnets into a peer-to-peer, homogeneous network architecture. Inspired by Tarzan's *mimics* system, we conducted a thorough analysis showing that, by introducing dummy messages and controlling the nodes' traffic rates, it is possible to prevent the detection of end-sending and end-receiving activities, even from a global network observer and collusions of corrupted nodes. We additionally proposed a new way to manage anonymous network identities, showing how relationship pseudonyms can allow end-receivers to remain anonymous even to end-senders. The protocol additionally proposes a cryptographically secure implementation of these pseudonyms, leveraging the homomorphic properties of the Elgamal scheme. Finally, the protocol was designed without anchoring trust into a particular central authority or a group of central servers. Indeed, privacy stems not from such central entities, but from the willingness of nodes to help each other in staying anonymous. By design, the more a node helps its neighbors with cover traffic, the more those can help it in return.

The resulting protocol is complex, and mainly aimed at users willing to pay a high price in efficiency and latency to obtain very strong privacy guarantees. The protocol is also less flexible than *plug'n'play* protocols that build circuits *on demand*, such as Tor, which is now bundled in browsers and allows users to start using the network immediately after joining it. Indeed, in the proposed protocol, before starting communications, a user joining the network must make itself known, and learn about other nodes in the network using *route proposals*. Additionally, every (oriented) communication session must be preceded by an *initialisation* requiring a secret token from the end-receiver. On the other hand, in contrast to protocols constructing circuits *on demand*, the proposed approach allows to build circuits shared by many senders, thus participating in preventing traffic analysis and ultimately ensuring strong sender anonymity.

In the next chapters, the security of the cryptographic components of the protocol is formally proven, and its practical efficiency studied.

5. Security and Privacy Proofs

The presented protocol invokes the security properties of various tools, and makes security claims, but in an informal way. This chapter gathers all the formal treatment of the protocol and its properties. Namely, it formally studies the security and privacy guarantees of the protocol, according to the principles of *provable security*.

In a first time, Section 5.1 introduces the relevant cryptographic proof frameworks (mainly, the UC and AnoA frameworks), and presents the approach and methodology to carry out our formal analysis of the protocol. Section 5.2 then informally summarises the results of the analysis. Then, Section 5.3 presents the formal security definition of the cryptographic schemes used in the protocol, such as the Elgamal scheme and hash functions. Finally, Sections 5.4, 5.5, and 5.6 respectively study the security of pseudonyms, of the route proposal mechanism and its properties, and of the whole protocol along with the sender anonymity (SA), receiver anonymity (RA), session unlinkability (SU), and message unlinkability (MU) properties. Each of these properties are formally defined, and proven. Only the traffic analysis resistance (TAR) property is not studied.

Due to the length of the proofs, this chapter provides only *proof sketches*, and the full proofs are placed in Appendix B.

5.1. General Methodology

This first section presents the cryptographic frameworks used as base for our analysis. It then discusses the difficulties encountered in the formal treatment of the protocol (and in particular, what *can* and *can not* be proved with the current state of provable security), and how we overcome them. Finally, this section summarises all the assumptions on which our analysis relies.

Table 5.1 describes the notations and symbols used throughout the chapter, that are not previously defined in this thesis.

Notation	Description	Example
$\overset{c}{\equiv}$	Computational Indistinguishability [Gol01]	$\{X(n)\}_{\forall n} \overset{c}{\equiv} \{Y(n)\}_{\forall n}$
$\mathcal{A}$	Cryptographic adversary	
$\mathcal{F}$	UC Ideal Functionality	
$\mathcal{S}\mathtt{im}$	UC Simulator	
$\mathcal{E}$	UC Environment	
Ch	AnoA Challenger	
$f(x,\cdot)$	Function with a fixed first argument	
$\mathcal{A}^f$	Oracle access to function f for the adversary	$\mathcal{A}^{\mathcal{O}}$, $\mathcal{A}^{f(x,\cdot)}$
Ω	Set of all nodes in the network	
Ω_c, Ω_h	Subset of corrupted (resp. honest) nodes	

Table 5.1. – Cryptographic Notations for Formal Proofs

5.1.1. Cryptographic Proof Frameworks

Following works on Tor [Bac+12; BMS16] or on the recent mixnet cMix [Cha+16] for instance, the proofs are made in two steps. The first step consists in proving that the protocol *realises* an *ideal functionality* using the UC framework [Can13]. In a second time, this ideal functionality is then further analysed, to prove properties such as SA and RA. This is done with the AnoA framework [Bac+13], or with a custom security definition depending on the property to prove.

5.1.1.a) The UC Framework

In provable security, the main way to formally express the security of a cryptographic *protocol* (as opposed to a *primitive*) is using the *real vs. ideal* paradigm. For an in-depth introduction to this concept, the reader may refer to a tutorial by Lindell [Lin16a]. The basic idea of this proof methodology is to prove that there exists no PPT *distinguisher* between a *real* execution in which an adversary $\mathcal{A}$ interacts with a protocol Π, and an

ideal execution in which a *simulator* $\mathcal{S}\texttt{im}$ (also called the *ideal adversary*) interacts with an *ideal functionality* $\mathcal{F}$. The latter is the *idealisation* of protocol Π, and is meant to capture its security properties. An ideal functionality $\mathcal{F}$ is usually devoid of cryptographic operations, and explicitly specifies the information that an actual adversary gets by interacting with the protocol in a real-world scenario.

The UC framework [Can13], is directly inspired from this proof methodology. Its specificity, compared to the so-called *standard model* [Lin16a], is to allow to prove security under *concurrent* composition of protocol instances. That is, while the standard model only guarantees security under *sequential* composition, a protocol proved secure with the UC framework ensures that, even if several protocol instances run concurrently, one protocol instance can not be used to attack another. To achieve *universal* composability, the UC framework transforms the *distinguisher* from the standard model into an *interactive distinguisher*. The latter is called the *environment*, and denoted $\mathcal{E}$. It is responsible for giving the protocol inputs to the adequate parties, and receives their outputs. It is not allowed to interact with the parties in any other way, meaning in particular that it can not play a corrupted party in the protocol. However $\mathcal{E}$ *controls* the adversary $\mathcal{A}$, which itself *can* interact with nodes and play a role in the protocol. $\mathcal{A}$ can be considered as $\mathcal{E}$'s proxy in the protocol. This seemingly trivial change to the standard model actually has crucial implications for the proofs. Indeed, while in the ideal execution in the standard model, the simulator is free to act as it will, in the ideal execution of the UC framework, the simulator must carry out the instructions of the environment just like $\mathcal{A}$ would (since otherwise, $\mathcal{E}$ would be trivially able to distinguish between the real and ideal executions). This can also be explained in terms of quantifiers: while in the standard model, the proofs must hold such that "for all $\mathcal{A}$, there exists a simulator $\mathcal{S}\texttt{im}$" that makes the ideal execution indistinguishable from the real one, in the UC framework the proof must hold such that "there exists a simulator $\mathcal{S}\texttt{im}$ such that for all $\mathcal{A}$". This implies in particular that in the UC framework, it is not possible to *rewind* of the adversary, a technique largely used *e.g.* in security proofs of zero-knowledge proofs schemes. Lindell's tutorial [Lin16a, Section 10.1] gives a good explanation of the differences between the standard model and the UC framework [Lin16a].

In addition to providing security under concurrent composition, the UC framework is very flexible. It allows to express many types of adversary (static, adaptive, semi-honest, or malicious, in particular), and many system models [CSV16]. It also comes with a *composition theorem* that allows a modular approach to proofs of complex systems: if a protocol uses one (or several) sub-protocols, the security of the sub-protocol(s) can be proved first and independently; and to prove the larger protocol, one can then safely use the sub-protocol's ideal functionality, arguably simpler than the corresponding sub-protocol. On the downside, the UC framework is also extremely complex. The interested reader may refer to the full paper [Can13], or to the simpler variant of the UC framework [CCL15], which was designed so as to make the UC framework easier to understand and use.

In addition to the elements already presented, the UC framework comes with specific terms, that we define here. The UC framework models parties of the protocol as instances of interactive turing machines (abbreviated ITI for *ITM Instance*), with input, output,

and communication tapes (the latter is dedicated to the receiving of protocol messages). The real and ideal executions are meticulously defined as an executing system of ITIs, with rules on which ITI can write on which tape of given ITIs. When an ITI writes in the input tape of another ITI and provides input to it, it is said to use the latter as *subroutine*. The real execution is denoted $\text{EXEC}^{\text{REAL}}_{\Pi,\mathcal{A},\mathcal{E}}$, and involves the environment $\mathcal{E}$, a protocol Π, and the adversary $\mathcal{A}$. The ideal execution is denoted $\text{EXEC}^{\text{IDEAL}}_{\mathcal{F},\mathcal{S}\texttt{im},\mathcal{E}}$, and involves $\mathcal{E}$, a simulator $\mathcal{S}\texttt{im}$ (also called the ideal adversary), and an ideal functionality $\mathcal{F}$. In the real execution, each party in Π is represented by one ITI, and exchanges messages with the other parties' ITIs. In the ideal execution, those parties are replaced with *dummy parties*, which are simple interface, that merely pass the inputs they receive from $\mathcal{E}$ to $\mathcal{F}$ and *vice-versa* (the dummy parties thus use the ideal functionality as subroutine). The information available to the real execution adversary $\mathcal{A}$, in the restricted model considered in this thesis, is basically all that is written on any tape of corrupted parties' ITIs, and all *messages* exchanged between ITIs (including those of honest parties). However, $\mathcal{A}$ can not see the (subroutine) input/outputs that parties give to other ITIs that they use as subroutine. The information available to the simulator $\mathcal{S}\texttt{im}$ in the ideal execution is all that is written on the tapes of corrupted (dummy) parties' ITIs, and what $\mathcal{F}$ explicitly *leaks*. However, $\mathcal{S}\texttt{im}$ does not see the inputs that honest dummy parties give to $\mathcal{F}$.

In the UC framework, a protocol Π is said secure if it *UC-realises* an ideal functionality $\mathcal{F}$, which corresponds to the requirement:

$$\exists \mathcal{S}\texttt{im} \text{ s.t. } \forall \mathcal{A},\ \left\{\text{EXEC}^{\text{REAL}}_{\Pi,\mathcal{A},\mathcal{E}}(z)\right\}_{\forall z\in\{0,1\}^*} \overset{c}{\equiv} \left\{\text{EXEC}^{\text{IDEAL}}_{\mathcal{F},\mathcal{S}\texttt{im},\mathcal{E}}(z)\right\}_{\forall z\in\{0,1\}^*}$$

where z represents the input that the environment $\mathcal{S}\texttt{im}$ receives (it can be considered as its source of randomness, from which $\mathcal{E}$ generates all other inputs). Finally, when, in a real execution, a sub-protocol Π_{sub} is replaced by a (sub-)ideal functionality $\mathcal{F}_{\text{sub}}$ used as subroutine by parties in the larger protocol, the proof is said to stand "in the $\mathcal{F}_{\text{sub}}$-hybrid model".

5.1.1.b) Analysis of the Ideal Functionality

Usually, proving that a protocol Π UC-realises an ideal functionality $\mathcal{F}$ is enough of a proof in itself. For *simple* and short protocols, $\mathcal{F}$ allows to immediately see what is learned by the adversary (since the information learned by the adversary is explicitly specified), and the protocol's security properties appear clearly. However, for network communication protocols (and more generally, for complex and large protocols), the ideal functionality often remains too large for the security properties of the protocol to be trivially deduced from it (although $\mathcal{F}$ is generally *simpler* than Π). Thus, it is common, at least in networking protocols, to further study the ideal functionality in order to put in evidence the security and anonymity properties of the protocol [Bac+12; BMS16; Cha+16]. Note that the analysis *could* be conducted directly on the protocol Π, thus skipping the UC framework step. However, ideal functionalities are usually simpler than the protocol they model, and being devoid of (almost) all cryptographic operations, allow for a crypto-free analysis.

The base idea to analyse an anonymous network protocol or a protocol's ideal functionality is to let the adversary choose two possible *neighboring* runs of the network. A random one among them is chosen, and the protocol is then run with $\mathcal{A}$ controlling the corrupted parties. Finally, $\mathcal{A}$ is asked to guess which of the two runs was picked [HM08; Bac+13]. For instance, for SA, the adversary would choose two different end-senders S_0 and S_1, a message m, and an end-receiver R. For a random $b \leftarrow\!\!\$\, \{0,1\}$, the sender S_b is then asked to send m to R. $\mathcal{A}$ must guess which of S_0 or S_1 sent the message. In our literature research, we found two existing tools that allow the formalisation of this security definition: the AnoA framework [Bac+13], and a framework by Hevia and Micciacio [HM08]. The latter does not take into account corrupted nodes, and expresses privacy w.r.t. *computational indistinguishability*. On the other hand, the AnoA framework models node corruption, and allows for a *quantitative* characterisation of anonymity, with possibly non-negligible advantage. Indeed, as pointed out by the authors of AnoA, in the presence of corrupted nodes, the adversary necessarily has a non-negligible advantage in breaking privacy properties. The Tor and cMix protocols were analysed with the AnoA framework [Bac+13; Cha+16]. Because we want to carry an analysis of the protocol in the presence of corrupted nodes, we also choose the AnoA framework.

More precisely, in this thesis, we use two methods. For high-level properties such as SA, RA, and SU, the AnoA framework [Bac+13] is used (as discussed next, for other properties we use a custom security definition). This framework uses security definitions inspired from the notion of differential privacy. Formally, in AnoA, a ideal function $\mathcal{F}$ (or, more generally, any kind of protocol) is said to be (ϵ, δ)-α-ind-cdp w.r.t. *adjacency function* α if for all PPT adversary $\mathcal{A}$,

$$\Pr\left[\mathcal{A}^{Ch(\mathcal{F},\alpha,0)} = 0\right] \leq e^{\epsilon} \cdot \Pr\left[\mathcal{A}^{Ch(\mathcal{F},\alpha,1)} = 0\right] + \delta \tag{5.1}$$

This equation formalises a setup in which $\mathcal{A}$ interacts with a challenger Ch (here, represented as an oracle of $\mathcal{A}$), which itself runs $\mathcal{F}$. The challenger is given as input the ideal functionality $\mathcal{F}$, the adjacency function α, and a bit $b \in \{0,1\}$. The function α models the security property to prove. For instance, the adjacency function for SA, in its most simplistic form, takes as input $r_0 = (S_0, R, m)$, $r_1 = (S_1, R, m)$ and b, and outputs (S_b, R, m). The terms r_0 and r_1 are called the *challenge rows*[1]. Ultimately, if the inequality (5.1) holds, this implies that $\mathcal{A}$ has only a *small* (but possibly non-negligible) probability of distinguishing a run with r_0 from a run with r_1. Finally, we note that the AnoA framework is flexible, and allows more complex α functions and various adversary models. We give more details on how the model considered in this thesis is formalised into the AnoA framework in Section 5.6.3.b (page 128), which deals with the proof of the SA, RA, and SU properties.

For route proposal properties, and for the MU property, we propose and use a custom security definition (which we describe in Section 5.5.3). Indeed, these properties can not be simply expressed with the AnoA framework. The next section discusses this point, and explains the rationale behind the custom security definition that we propose.

[1] The term *row* stems from the literature on differential privacy in databases.

5.1.2. Approach and Assumptions

With the presented tools, the definitions and proofs of security and anonymity properties are divided in the following way. Firstly, pseudonyms are proven secure, using traditional indistinguishability-based definitions. Secondly, the route proposal mechanism is studied. It is expressed as a protocol Π_{rtprop}, and shown to UC-realise an ideal functionality $\mathcal{F}_{\mathsf{rtprop}}$. The latter is then analysed to prove the *route proposal homogeneity*, *route proposal indistinguishability*, *propagation untraceability*, and *return tric untraceability* properties of the route proposal mechanism. (see Section 4.5.1.b, page 73). In a third time, the entire protocol Π is described as pseudo-code, using $\mathcal{F}_{\mathsf{rtprop}}$ as subroutine. It is then proved to UC-realise an ideal functionality $\mathcal{F}$. Finally, this latter functionality is analysed, to prove the SA, RA, SU, and MU properties. We choose this two-step analysis of the protocol (first, the route proposal mechanism Π_{rtprop}, and then the entire protocol Π) in order to be able to study the route proposal properties with $\mathcal{F}_{\mathsf{rtprop}}$, and to reduce the complexity of proving the whole protocol in one large proof of UC-realisation.

The remainder of this section details what can and can not be proved with the tools we consider to use, and how we work around the issues that arise. It also discusses our custom security definition, and details the assumptions on which all proofs rely.

5.1.2.a) What Can and Can Not be Proved

In the realm of formal proofs, it is widely admitted that, with today's knowledge, there are elements and properties of network communication protocols that *can not* be proved. That is, some elements of design, such as dummy messages or message re-ordering strategies, are difficult to take into account in formal proofs. There are also properties for which a proof methodology is yet to be discovered (if it exists).

The most straightforward example is the impossibility to prove resistance to traffic analysis [SW06]: it seems that any non-trivial protocol will always allow the adversary to trace messages, if given enough (polynomial) time. Some works choose to exclude these elements that are of a *non-cryptographic* nature from the model and the proofs, stating that *e.g.* traffic analysis attacks should be *handled by orthogonal mechanisms* [DG09]. Other works take a conservative approach (as it is standard in provable security), and consider the *worst-case* assumption. That is, they make the assumption that the adversary is able to perform traffic analysis, and in particular, can perfectly trace message. It is the case of the formal treatment of onion routing [CL05] and of the Tor protocol [Bac+12].

In this thesis, the security of the protocol relies largely on the impossibility to perform traffic analysis (contrarily to low latency protocols). Ensuring TAR is the role of the message re-ordering, dummy messages, and controlled traffic rates mechanisms. However, in light of the above remarks, it is not possible to formally prove that these mechanism achieve the desired level of protection. As a result, if we were to choose a *conservative* approach, we would assume that they do not offer any protection at all. Therefore, the global network observer in our considered adversary model would be able to perfectly trace all messages, which amounts to stating that the protocol provides no anonymity whatsoever. On the other hand, if we were to choose to make the (strong) assumption

that the protocol perfectly resists traffic analysis, we obtain security guarantees that are possibly too *optimistic*. For the sake of obtaining proofs as close to the actual anonymity provided by the protocol, we propose a middle ground between these two extremes. Namely, we assume that internal adversaries *can* perform traffic analysis, but not external ones. This boils down to assuming that the dummy messages and controlled traffic rates mechanisms of the protocol prevent network observers from distinguishing real from dummy messages, and thwarts the observation of messages exchanged between neighbors. This chosen approach translates into an assumptions, formalised in the UC framework under an ideal functionality $\mathcal{F}_{\mathsf{link}}$, presented in the next section.

This settles the question of TAR. Another element that arguably participates in the security of the protocol is the *concurrency* among network events, *e.g.* among route proposals, or oriented communications. Informally, the fact that several events happen in parallel in the network provides *cover traffic*, and introduces uncertainty in the adversary's observation. However, this element can not be included in the formal proofs either. To illustrate this issue, the most straightforward example is that of the route proposal homogeneity property, which states that it should be impossible to distinguish a self-proposal from a relayed one. This property is formalised by running the network either with Y self-proposing to a (possibly corrupted) proposee X, either with Y proposing to X a route towards R. Let us assume in a first time, for the sake of the argument, that there is no concurrency among network event, as it is the case for the analysis of Tor with the AnoA framework [Bac+13], and thus there is only one route proposal occurring in the network at any time. Then, distinguishing the self-proposal from the relayed one is trivial for the adversary, since only the latter necessitates a return trip. Indeed, if, during the challenge route proposal, the adversary sees that one of its corrupted nodes Z is solicited as part of a return trip, it can be sure that a *relayed* proposal was executed. Now, let us assume that there are always several route proposals being carried out at the same time, which can also be modeled in the AnoA framework, and which better reflects a real-world scenario. Then the situation is not as favorable to the adversary. Indeed, if, during the challenge, a corrupted node Z is solicited as part of a return trip, she can not know for sure whether this return trip relates to the challenge route proposal, or to some other route proposal happening in parallel. In a thorough proof, this *uncertainty* of the adversary should be quantified, or at least over-approximated. However, there is no formal foundations for such an analysis, not in AnoA, nor in any other existing framework, to our knowledge.

To overcome this difficulty, we choose to largely over-approximate the adversary's advantage: since the impact of the presence of one (or several) corrupted nodes can not be accurately quantified, it is considered that it fully breaks the property (*i.e.* that it gives a probability 1 for the adversary to break the property). For instance, for RA, we consider that, if there is a corrupted node on the second leg of the challenge session, the adversary finds the receiver with probability 1. The benefit of this approach is that it yields results that hold even against a very strong adversary, able to control all the traffic in the network (*i.e.* even when there is no concurrency among network events). On the other hand, this only gives a lower bound on the actual anonymity provided by

the protocol.

On top of these difficulties, the MU property, as well as the *return trip untraceability* property of route proposals, can not simply be expressed with the AnoA framework (and any other existing framework, to our knowledge). Indeed, AnoA is mainly suited to express *high level* properties, pertaining to communication sessions (such as SA, RA, and SU), but not to express more fine-grained properties that necessitate to formulate challenges that do not apply on full sessions, but on messages, or parts of routes.

Another (orthogonal) issue arises for the formalisation of the route proposal properties. Indeed, even if we set aside the issue of corrupted relay nodes in the above example about the route proposal homogeneity property, there is another pitfall. For instance, the adversary may know, from past interactions in the network, that the corrupted proposee X, before the challenge, has n_1 routes towards Y, and $n_2 \neq n_1$ routes towards R. Thus, at the end of the challenge, when X learns either $PS = PS_{X \to Y}$ or $PS = PS_{X \to R}$, X can see that it previously had n_1 or n_2 routes towards this pseudonyms PS, and deduce whether Y was self-proposing or not. More generally, route proposal properties must be studied by taking into account all the previous route proposal the adversary was involved in, and what she has learned through them. We do not know of a way to take into account all past actions of the adversary, nor the *side-channel* information in the example of route proposal homogeneity. Therefore, in a first step, we aim at proving these properties *outside* of the network dynamics, and without taking these elements into account.

In light of these remarks, we propose in Section 5.5.3 a custom security definition based on adversarial *views*, and use it to prove the MU property, and the properties of route proposals. This definition is designed to address all the above mentioned issues. That is, it circumvents the issue regarding the corrupted relay nodes' impact on the adversary's advantage (by allowing to model challenges in which the corrupted relay nodes are the same in both cases, *i.e.* whether $b = 0$ or $b = 1$). Secondly, it allows the expression of more *fine-grained* challenges, on *portions of routes* rather than on entire communications. Finally, it allows to analyse a route proposal *out of the dynamics of the network*, thus avoiding the need to take into account all the information that the adversary may have obtained from past route proposals.

5.1.2.b) Assumptions

Here, we summarise all the cryptographic and network assumptions made throughout the chapter. Generally, the adversary is considered PPT, and can eavesdrop *all* communication links, and/or corrupt an arbitrary fraction of the network. The adversary is considered passive (or *semi-honest*), and static (as opposed to adaptive), meaning that the set of corrupted node is fixed at the beginning of the network lifetime and does not change afterwards. Appendix B.2 expands on how these adversary models translate into the UC framework. Our results rely on the assumption that the pseudonym indistinguishability property, and the IND-CPA, IK-CPA, and USS properties of the Elgamal schemes hold. Ultimately, this means that our proofs rely on the hardness

assumption of the DDH problem, and on assumption that the hash function used is indistinguishable from a random function. All these cryptographic properties and hard problems are presented in Section 5.3.

Additionally, when proving the security of the full protocol, with oriented communications, it is assumed that end-senders and indirection nodes do not collude, so that the adversary does not learn the *dst* value of honest nodes. Also, it is assumed that route proposals essentially consist in the computation of a pseudonym, and do not leak information such as the route length or identity of the nodes on the route. Without loss of generality, it is also assumed that there is an upper bound l_{max} on the length of the created route.

The assumption presented in the previous section, on the impossibility for external adversaries to perform traffic analysis, translates into an ideal functionality $\mathcal{F}_{\mathsf{link}}$ in the UC framework. This ideal functionality abstracts the dummy messages policy, the controlled traffic rate, and the message re-ordering mechanisms. That is, in the real UC execution, nodes do not exchange messages, but only communicate by input/outputs, using $\mathcal{F}_{\mathsf{link}}$ as subroutine. As a result, UC proofs are said to stand *in the* $\mathcal{F}_{\mathsf{link}}$*-hybrid model*. This assumption itself is studied in Appendix B.7, which presents a protocol that arguably UC-realises $\mathcal{F}_{\mathsf{link}}$ under strong assumptions on the traffic load. On the other hand, it is assumed that corrupted nodes can perform traffic analysis. For instance, on a route of the following form:

$$(S \overset{cid_1}{\longrightarrow} Z_1 \overset{cid_2}{\longrightarrow} Z_2 \overset{cid_3}{\longrightarrow} Z_3 \overset{cid_4}{\longrightarrow} Z_4 \overset{cid_5}{\longrightarrow} R)$$

if Z_1 and Z_4 are corrupted, it is assumed that they are able to know that they are on the same route. That is, when Z_1 sends a message to Z_2 with cid_2, it knows that the message will arrive to Z_4 from Z_3 and with cid_4. Note however that, although the receiver R is situated just after the corrupted node Z_4, this does not mean, in the general case, that Z_4 knows that R is the receiver: for all it knows, R could be another relay towards a receiver further down the road.

5.2. Summary of Results

This section summarises the results of the formal analysis of the protocol. First, we show that the route proposal mechanism UC-realises the ideal functionality $\mathcal{F}_{\mathsf{rtprop}}$ (defined in Fig. 5.6 on page 110), and then that the full protocol UC-realises the ideal functionality $\mathcal{F}$ (defined in Fig. 5.10 on page 125). Without further analysis, a simple inspection of $\mathcal{F}_{\mathsf{rtprop}}$ and $\mathcal{F}$ immediately shows that external adversaries do not get any information whatsoever about route proposals nor about oriented communications. This is in particular due to the fact that proofs stand in the $\mathcal{F}_{\mathsf{link}}$-hybrid model. Therefore, we are able to show that all properties (SA, RA, SU, MU, and route proposal properties) perfectly hold against any PPT external adversary.

Against corrupted nodes (*i.e.* internal adversaries), the situation is more complex. Firstly, it can be noted that both ideal functionalities leak dst^{src} values to end-senders and/or end-receivers. This is due to the way pseudonyms are computed. Indeed, $\mathcal{F}_{\mathsf{rtprop}}$

implicitly contains the sub-protocol from Fig. 4.5 (on page 75) that computes the pseudonyms during route proposals; and, similarly, $\mathcal{F}$ implicitly contains the sub-protocol from Fig. 4.8 (on page 85) that computes the pseudonyms during oriented communication initialisations. The ideal functionalities show that, during a route proposal, a (corrupted) end-receiver R learns ${dst_R}^{src_X}$ w.r.t. proposee X, and during an oriented communication initialisation, a (corrupted) end-sender S learns ${dst_R}^{src_I}$ w.r.t. the end-receiver R and the indirection node I. Another remark that can be made by inspecting the ideal functionalities, is that (corrupted) relay nodes do not learn information from the link messages they relay. More exactly, they do not learn information from the messages *themselves*. However, the very fact that a corrupted node is solicited to relay a message, on a specific route (with a specific *cid* value and next hop node) does indirectly reveal information. As already discussed in the previous section, quantifying the advantage this information provides to the adversary is far from trivial, and not possible with the tools at hand.

These general remarks do not however prevent the proofs of the protocol properties to be carried out, since we work around these shortcomings. With the approach and assumptions described in Section 5.1.2, the proof of each property results in a quantification of the probability that the adversary breaks that property. This probability depends in particular on the number of corrupted nodes in the network. Namely, the result for each property is as follows:

- Sender anonymity (SA) holds with probability equal to $\binom{|\Omega|-l_{max}}{|\Omega_c|}/\binom{|\Omega|}{|\Omega_c|}$, which corresponds to the probability that the first leg of the oriented communication is devoid of corrupted nodes.
- Receiver anonymity (RA), holds with the same probability as SA (up to a negligible additive factor, however), which corresponds to the probability that the *second* leg of the oriented communication is devoid of corrupted nodes.
- Session unlinkability (SU) holds with probability $\binom{|\Omega|-2l_{max}+1}{|\Omega_c|}/\binom{|\Omega|}{|\Omega_c|}$, which corresponds to the probability that both legs are devoid of corrupted nodes.
- Message unlinkability (MU) is divided into MU-session and MU-tracing, as defined in Chapter 1 (on page 13). We do not make an attempt at proving MU-session, because it seems that it is trivially broken by the adversary. More accurately, we were not able to find a meaningful formal definition of MU-session that is not trivially broken by $\mathcal{A}$. To prove MU-tracing, we express it in two ways, depending on whether the challenge consists in tracing messages on the first or second leg of an oriented communication route. It appears that MU-tracing for the second leg holds perfectly (up to a negligible probability), and with probability $\binom{|\Omega|-2}{|\Omega_c|}/\binom{|\Omega|}{|\Omega_c|}$ for the first leg. This discrepancy is due to the privileged place that the indirection node occupies in oriented communications.
- Finally, the four route proposal properties are proven to hold perfectly (up to a negligible probability).

In these results, every *negligible factor* that appears is due to the leaking of pseudonyms or encryptions of one to the adversary. In all cases, these informations give a *negligible* advantage $\mathsf{negl}(\lambda)$, by the indistinguishability of pseudonyms and the IK-CPA property of the Elgamal scheme. Additionally, note that we are able to show that the route proposal properties hold *perfectly* only because we formally define these properties in a way that nullifies the impact of corrupted nodes, and because our custom security definitions takes route proposals *out of context*. Although this approach significantly reduces the impact of the results, this is a first step towards proving these properties in more general cases.

In a network with $|\Omega| = 1000$ nodes and a maximum route length of $l_{max} = 10$ hops, these results lead to the following probabilities for SA, RA, SU, and MU-tracing to hold. When there are $|\Omega_c| \,/\, |\Omega| = 1\%$ of corrupted nodes in the network, SA and RA hold with probability 0.9, SU holds with probability 0.83, and MU-tracing (on the first leg) holds with probability 0.98. However, these figures decrease rapidly when the ratio of corrupted nodes augments. Indeed, when $|\Omega_c| \,/\, |\Omega| = 10\%$, SA and RA hold only with probability 0.35, and SU with probability 0.13. However, these results compare well with those of a recent analysis of the Tor network [BMS16]. The authors of the study show that, with only 20 corrupted Tor servers among the 6000 ones in the network at the time [2], which means a corruption ratio of $c/n \approx 0.33\% < 1\%$, SA and RA respectively hold with probability ≈ 0.85 and ≈ 0.75. Chapter 6 further illustrates these results, and completes them with an empirical quantification of anonymity, inspired from a (non-formal) methodology proposed by the authors of Tarzan.

5.3. Formal Security Definition of Cryptographic Assumptions

Before describing the proofs, the cryptographic assumptions and hard problems used in this work must be formally defined. In particular, the definitions of the semantic security (IND-CPA), key-privacy (IK-CPA), and universal semantic security (USS) properties are presented. Recall that the group $\mathbb{G}$ is defined for primes p and q such that $p = 2q + 1$, as the subgroup of $\mathbb{Z}_i{}^*$ of order q. Appendix A provides more details on the construction of $\mathbb{G}$.

5.3.a) The Decisional Diffie-Hellman and Discrete Logarithm Problems

The DDH problem consists, given $g, g^a, g^b, g^c \in \mathbb{G}^4$, with $a, b \leftarrow\!\!\$\, \mathbb{Z}_q$, in distinguishing whether $c = ab$ or $c \leftarrow\!\!\$\, \mathbb{Z}_q$. Solving this problem in polynomial time is believed impossible in the subgroup $\mathbb{G}$ considered in this work. The hardness assumption of the DDH problem is denoted in short as the *DDH assumption*.

[2] https://metrics.torproject.org/networksize.html

5.3.b) IND-CPA Security of PKE Schemes

Semantic security is one of the most basic security definitions in cryptography. The Elgamal scheme is proven IND-CPA under the DDH assumption [TY98]. The generic, game-based indistinguishability definition is as follows.

Definition 20 (Indistinguishability under Chosen Plaintext Attacks (IND-CPA)). *Let* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a PKE scheme, and* $\lambda \in \mathbb{N}$. *The scheme is said to ensure the IND-CPA property if the function* $\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A}}(\lambda)$ *is negligible for any PPT adversary* $\mathcal{A}$, *where*

$$\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A}}(\lambda) := \left|\Pr\left[\mathbf{Exp}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A}}(\lambda) = 1\right] - 1/2\right| \tag{5.2}$$

with $\mathbf{Exp}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A}}(\lambda)$ *depicted in Fig. 5.1.*

$\mathbf{Exp}^{\mathsf{ind\text{-}cpa}}_{\mathcal{A}}(\lambda)$

$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, pp)$; $b \leftarrow\!\!\$\ \{0,1\}$
$(st, m_0, m_1) \leftarrow \mathcal{A}(pk)$
$b^* \leftarrow \mathcal{A}(st, \mathsf{Enc}(pk, m_b))$
return $b = b^*$

$\mathbf{Exp}^{\mathsf{ik\text{-}cpa}}_{\mathcal{A}}(\lambda)$

$pp \leftarrow \mathsf{Setup}(1^\lambda)$; $b \leftarrow\!\!\$\ \{0,1\}$
$(pk_0, sk_0) \leftarrow \mathsf{KeyGen}(1^\lambda, pp)$
$(pk_1, sk_1) \leftarrow \mathsf{KeyGen}(1^\lambda, pp)$
$(st, m) \leftarrow \mathcal{A}(pk_0, pk_1)$
$b^* \leftarrow \mathcal{A}(st, \mathsf{Enc}(pk_b, m))$
return $b = b^*$

$\mathbf{Exp}^{\mathsf{uss}}_{\mathcal{A}}(\lambda)$

$pp \leftarrow \mathsf{Setup}(1^\lambda)$; $b \leftarrow\!\!\$\ \{0,1\}$
$(pk_0, sk_0) \leftarrow \mathsf{KeyGen}(1^\lambda, pp)$
$(pk_1, sk_1) \leftarrow \mathsf{KeyGen}(1^\lambda, pp)$
$(st, m_0, m_1, (r_0, r_{\mathrm{one}0}), (r_1, r_{\mathrm{one}1})) \leftarrow \mathcal{A}(pk_0, pk_1)$
For $i \in \{0,1\}$:
  $c_i \leftarrow \mathsf{Enc}(pk_i, m_i, r_i), c_{\mathrm{one}i} \leftarrow \mathsf{Enc}(pk_i, 1, r_{\mathrm{one}i})$
  $c'_i \leftarrow \mathsf{ReEnc}_{\mathrm{nopk}}(c_{\mathrm{one}i}, c_i)$
$b^* \leftarrow \mathcal{A}(st, c_b, c_{1-b})$
return $b = b^*$

Figure 5.1. – IND-CPA, IK-CPA, and USS Security Games

5.3.c) Key-Privacy of PKE Schemes

Key-privacy [Bel+01] is a property of to PKE scheme which is orthogonal to IND-CPA security. Abstractly, ciphertexts of a scheme satisfying key-privacy do not leak information about the public key under which they are encrypted. This property only makes sense for public keys that are based on the same *public parameters*. For instance, in the Elgamal scheme, that would be public keys belonging to the same group $\mathbb{G}$. Thus, to formulate key-privacy, we add Setup, a fourth operation to the description of PKE schemes, that inputs 1^λ and outputs public parameters pp, that are then fed to KeyGen[3]. Generally, pp is assumed to be publicly known to any entity in the system. The general definition of key-privacy, formally denoted IK-CPA, is as follows.

[3] In a scheme description without the Setup operation, KeyGen implicitly generates its own public parameters *on-the-fly*.

Definition 21 (Indistinguishability of Keys under Chosen Plaintext Attacks (IK-CPA)). *Let* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a PKE scheme with a common key generation procedure* Setup*, and let* $\lambda \in \mathbb{N}$*. The scheme is said to ensure the IK-CPA property if the function* $\mathsf{Adv}^{\mathsf{ik\text{-}cpa}}_{\mathcal{A}}(\lambda)$ *is negligible for any PPT adversary* $\mathcal{A}$*, where*

$$\mathsf{Adv}^{\mathsf{ik\text{-}cpa}}_{\mathcal{A}}(\lambda) \coloneqq \left| \Pr\left[\mathbf{Exp}^{\mathsf{ik\text{-}cpa}}_{\mathcal{A}}(\lambda) = 1 \right] - 1/2 \right| \tag{5.3}$$

with $\mathbf{Exp}^{\mathsf{ik\text{-}cpa}}_{\mathcal{A}}(\lambda)$ *depicted in Fig. 5.1.*

5.3.d) Universal Semantic Security of URE

Universal semantic security informally states that a ciphertext can not be recognised after it has been re-encrypted. In the case of the Elgamal scheme for instance, universal semantic security actually stems from the IND-CPA and IK-CPA properties. The *security game* $\mathbf{Exp}^{\mathsf{uss}}_{\mathcal{A}}(\lambda)$ defining USS actually look like a merger of the IND-CPA and IK-CPA games. Here, the notion is presented according to the definition by Golle *et al.* [Gol+04], but specifically adapted to the Elgamal scheme.

Definition 22 (Universal Semantic Security under Re-Encryption (USS)). *Let* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReEnc}_{\mathtt{nopk}})$ *be the Elgamal PKE scheme augmented with the* $\mathsf{ReEnc}_{\mathtt{nopk}}$ *operation (defined in Section 4.3.2), and let* $\lambda \in \mathbb{N}$*. The Elgamal scheme is said to ensure the USS property if the function* $\mathsf{Adv}^{\mathsf{uss}}_{\mathcal{A}}(\lambda)$ *is negligible for any PPT adversary* $\mathcal{A}$*, where*

$$\mathsf{Adv}^{\mathsf{uss}}_{\mathcal{A}}(\lambda) \coloneqq \left| \Pr\left[\mathbf{Exp}^{\mathsf{uss}}_{\mathcal{A}}(\lambda) = 1 \right] - 1/2 \right| \tag{5.4}$$

with $\mathbf{Exp}^{\mathsf{uss}}_{\mathcal{A}}(\lambda)$ *depicted in Fig. 5.1.*

5.3.e) Hash Functions

The Keccak function family, and in particular the SHA-3 hash function, has its security defined by comparison to a truly random function $\mathtt{Rand}$, which returns truly random number (but always the same number for the same input). Defining the security of hash functions in this way is different from modeling the hash function as a *random oracle*. Indeed, contrarily to the *random oracle model* (ROM), here, the function $\mathtt{Rand}$ is directly accessible to the adversary, and can not be *programmed* in reduction proofs [Fis+10], thus better modeling real-world setups.

The authors of Keccak define the security of their function in the following way (adapted from [Ber+11])

Definition 23 (Hash Function Indistinguishable from a Random Function). *A hash function* $\mathsf{h} : \{0,1\}^* \to \{0,1\}^n$ *is indistinguishable from a random function* $\mathtt{Rand}$ *if the function* $\mathsf{Adv}^{\mathsf{h}/\mathtt{Rand}}_{\mathcal{A}}(\lambda)$ *is negligible for any PPT adversary* $\mathcal{A}$*, where*

$$\mathsf{Adv}^{\mathsf{h}/\mathtt{Rand}}_{\mathcal{A}}(\lambda) \coloneqq \left| \Pr\left[\mathcal{A}^{\mathcal{O}}(\lambda) \to 0 \;\middle|\; \mathcal{O} = \mathsf{h} \right] - \Pr\left[\mathcal{A}^{\mathcal{O}}(\lambda) \to 0 \;\middle|\; \mathcal{O} = \mathtt{Rand} \right] \right|$$

The above definition is used to prove a part of the security of pseudonyms. However, the more standard notions of preimage and collision resistance are also used, and presented below for the sake of completeness. Note that the above definition of indistinguishability from a random function implies all three following properties (definitions are adapted from [RS04]):

Definition 24 (Hash Function Properties).

Preimage resistance

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{h}-\mathsf{pre}}(\lambda) := \Pr\left[\mathsf{h}(x) = \mathsf{h}(x') \;\middle|\; x \leftarrow_{\$} \{0,1\}^*; x' \leftarrow \mathcal{A}(1^\lambda, \mathsf{h}(x))\right] \leq \mathsf{negl}(\lambda)$$

2nd preimage resistance

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{h}-2^{nd}\mathsf{pre}}(\lambda) := \Pr\left[x \neq x' \wedge \mathsf{h}(x) = \mathsf{h}(x') \;\middle|\; x \leftarrow_{\$} \{0,1\}^*; x' \leftarrow \mathcal{A}(1^\lambda, x)\right] \leq \mathsf{negl}(\lambda)$$

Collision resistance

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{h}-\mathsf{coll}}(\lambda) := \Pr\left[x \neq x' \wedge \mathsf{h}(x) = \mathsf{h}(x') \;\middle|\; (x, x') \leftarrow \mathcal{A}(1^\lambda)\right] \leq \mathsf{negl}(\lambda)$$

5.3.f) Secret Sharing

This thesis makes use of a custom and extremely simple secret sharing scheme, which was already presented in Chapter 4. It consists, given $e \in \mathbb{G}$, in splitting e into two shares $sh_1 \leftarrow_{\$} \mathbb{G}$ and $sh_2 = e/sh_1$. The reconstruction of e from the shares involves a single group multiplication.

Adapting the definition from generic secret sharing scheme [MOV96], the present scheme is said *secure* if given only sh_1 or only sh_2, no information (in the sense of information theory) is learned about e. Clearly, this scheme is correct and secure w.r.t. the above definition since in $\mathbb{G}$, for any given e_1, $\{e_1 \cdot e_2 \mid e_2 \in \mathbb{G}\} = \mathbb{G}$, and thus given one share and no information on the other, the secret could be any group element.

5.4. Security of Pseudonyms

The first protocol component that is studied in this chapter are the pseudonyms. As explained in Section 4.5.2, the pseudonyms must ensure the *uniqueness*, *one-wayness*, and *indistinguishability* properties. This section defines them formally and proves that our implementation of pseudonyms guarantees them. Note that the security of the *protocols* computing the pseudonyms (*e.g.* during a route proposal) is not treated in this section, but in following ones.

The following theorem defines the three properties for the proposed pseudonym implementation.

Theorem 1 (Security of Pseudonyms). *Let $\mathbb{G}$ be a publicly known group where the DDH assumption holds, and $\mathsf{h} : \mathbb{G} \to \{0,1\}^n$ be a public hash function satisfying Definition 23 with n polynomial in λ. Denote by $f : \mathbb{Z}_q^* \times \mathbb{G} \to \{0,1\}^n$ the function computing the pseudonyms,* i.e. *which on input (src, dst) returns $\mathsf{h}(dst^{src})$.*

The following properties hold for any PPT adversary $\mathcal{A}$ with direct (public) access to $f(\cdot,\cdot)$:

– Uniqueness*:* $\forall\ src \in \mathbb{Z}_q*$,

$$\mathsf{Adv}^{\mathsf{ps-uniq}}_{\mathcal{A}}(\lambda) \coloneqq \Pr[f(src, dst_1) = f(src, dst_2) \mid dst_1, dst_2 \leftarrow_{\$} \mathbb{G}\,] \leq \mathsf{negl}(\lambda)$$

– One-wayness*:* $\forall\ src \in \mathbb{Z}_q^*$,

$$\mathsf{Adv}^{\mathsf{ps-ow}}_{\mathcal{A}}(\lambda) \coloneqq \Pr\left[f(src, \mathcal{A}^{f(\cdot,dst)}(1^\lambda, src)) = f(src, dst) \;\middle|\; dst \leftarrow_{\$} \mathbb{G}\right] \leq \mathsf{negl}(\lambda)$$

– Indistinguishability*:* $\forall\ src \in \mathbb{Z}_q^*$

$$\mathsf{Adv}^{\mathsf{ps-ind}}_{\mathcal{A}}(\lambda) \coloneqq \left|\Pr\left[\mathbf{Exp}^{\mathsf{ps-ind}}_{\mathcal{A},src}(\lambda) = 1\right] - 1/2\right| \leq \mathsf{negl}(\lambda)$$

with $\mathbf{Exp}^{\mathsf{ps-ind}}_{\mathcal{A},src}$ *defined in Fig. 5.2.*

$\mathbf{Exp}^{\mathsf{ps-ind}}_{\mathcal{A},src}(\lambda)$

$dst_0, dst_1 \leftarrow_{\$} \mathbb{G}$; $b \leftarrow_{\$} \{0,1\}$
Let $PS_1 \coloneqq f(src, dst_b)$ and $PS_2 \coloneqq f(src, dst_{1-b})$
$b^* \leftarrow \mathcal{A}^{f(\cdot,dst_0), f(\cdot,dst_1)}(1^\lambda, src, PS_1, PS_2)$
return $\perp$ **if** $f(src, dst_0)$ or $f(src, dst_1)$ was called by $\mathcal{A}$
Otherwise, **return** $b^* = b$

Figure 5.2. – Pseudonym Indistinguishability Security Game

Intuitively, uniqueness states that, for any nodes X, R, R' such that $R \neq R'$, $PS_{X\to R}$ is different from $PS_{X\to R'}$ with all but negligible probability. This ultimately prevents X from mistaking an end-receiver for another, and ensures the good functioning of the routing in the network. One-wayness ensures that dst_R can not be recovered from a pseudonym, and ultimately that a node can not impersonate another node (as end-receiver). Indistinguishability implies that a pseudonym does not leak any information on the end-receiver it designates, or rather on the dst value of the end-receiver it designates.

The one-wayness property is modeled by giving $\mathcal{A}$ oracle access to $f(\cdot, dst)$, allowing her to compute pseudonyms for the challenge dst value. Note that this modeling is stronger than simply giving $\mathcal{A}$ one pseudonym $f(src, dst)$. This accounts for the fact that several corrupted nodes can collude to attack one specific pseudonym, or equivalently that the adversary can have polynomially many pseudonyms that she knows designate the same end-receiver. This is a worst-case scenario, since it implicitly assumes that corrupted nodes in the collusion were *already* able to link these pseudonyms together in some way. Likewise, in the pseudonym indistinguishability game, $\mathcal{A}$ has access to two oracles $f(\cdot, dst_0)$ and $f(\cdot, dst_1)$, to represent a scenario in which each corrupted

node $X_i \in \{X_1, \ldots, X_n\}$ (for a polynomial n) already know that $f(src_{X_i}, dst_0)$ and $f(src_{X_i}, dst_1)$ designates respectively R_0 and R_1. The definition states that, even in this case, if a new corrupted node X_{n+1} comes in with $f(src, dst_0)$ and $f(src, dst_1)$, it can not know which value designates which end-receiver. Finally, note that the properties hold for all src value in $\mathbb{Z}_q^*$, meaning that any src value $\mathcal{A}$ can choose will not give her a better than negligible advantage.

Proof Sketch 1 (Theorem 1). For a full proof, see Appendix B.1. Each property is proven independently. Although all three properties can be proved under the assumption that h *is indistinguishable from* `Rand`*, when possible, we simply considered it as a hash function with e.g. collision resistance. The versatility of the Keccak function family allows this.*

- *Uniqueness is trivially proven based on the assumption that the hash function used is collision resistant.*
- *We show that if there exists an adversary $\mathcal{A}$ successfully outputting $dst_{\mathcal{A}}$ such that $f(src, dst_{\mathcal{A}}) = f(src, dst)$, then it is possible to construct an adversary $\mathcal{B}$ that distinguishes* h *from a random function* `Rand`*. Note that intuition would suggest that one-wayness can be proved from assuming the preimage and 2^{nd} preimage resistance of the hash function, the reduction fails because it is not possible to construct an adversary $\mathcal{B}$ that successfully binds its own challenge $\mathsf{h}(x)$ with $\mathcal{A}$'s and at the same time answer $\mathcal{A}$'s oracle queries consistently. This is because the proof does* not *take place in the random oracle model. Alternatively, it is possible to prove that indistinguishability implies one-wayness.*
- *Indistinguishability is trivially proven under the assumption that the hash function is indistinguishable from a random one. Intuitively, pseudonym indistinguishability holds because dst^{src} is passed through a function that destroys all algebraic properties between (pairs of) pseudonyms.*

5.5. Security of the Route Proposal Mechanism

In Section 4.5.1.b, four properties for the route proposal mechanism were put forward: *route proposal homogeneity*, *route proposal indistinguishability*, *untraceable route proposal propagation*, and *untraceable return trip*. The present section defines them more formally, and proves them. First, the topology dissemination phase is given in pseudo-code Π_{rtprop}, then modeled as a UC ideal functionality $\mathcal{F}_{\mathsf{rtprop}}$. Then, Π_{rtprop} is shown to UC-realise $\mathcal{F}_{\mathsf{rtprop}}$. Finally, based on a study of the information an adversary gets from $\mathcal{F}_{\mathsf{rtprop}}$, the four properties are proved with a custom security definition.

5.5.1. Modeling Π_{rtprop} into an Ideal Functionality $\mathcal{F}_{\mathsf{rtprop}}$

Prior to describing Π_{rtprop}, we present the two ideal functionalities that it uses as subroutines: $\mathcal{F}_{\mathsf{link}}$ and $\mathcal{F}_{\mathrm{reg}}$. They are respectively described in Figures 5.3 and 5.4. Their pseudo-code, and more generally, the pseudo-code of all protocols and ideal functionalities

in this chapter, is written in the message-state paradigm, as it is standard for describing network protocols in the UC framework [Bac+12; Cha+16]. That is, pseudo-codes in this chapter feature *entry points* corresponding to the receiving of messages, inputs, or subroutine outputs. These entry points are marked by the **upon** keyword.

1 : The functionality $\mathcal{F}_{\mathsf{link}}$ is responsible for delivering link messages.
2 : **upon** input $(\mathsf{link\text{-}send}, sid, Y, m)$ from party X:
3 : Output $(\mathsf{link\text{-}rcvd}, sid, X, m)$ to party Y.

Figure 5.3. – The Link Message Functionality $\mathcal{F}_{\mathsf{link}}$

1 : The functionality $\mathcal{F}_{\text{reg}}$ is responsible for delivering key pairs consistently across all components of the protocol.
2 : **upon** input (keys, sid, X) from party P:
3 : **if** $T[X] = \perp$ **then** set $T[X] \leftarrow \mathsf{KeyGen}(\lambda)$.
4 : **if** $P = X$ **or** (P is the adversary **and** $X \in \Omega_c$) **or** P is $\mathcal{F}_{\mathsf{rtprop}}$ **then**
5 : Output $T[X]$ to party P.

Figure 5.4. – The Register Functionality $\mathcal{F}_{\text{reg}}$

The functionality $\mathcal{F}_{\mathsf{link}}$ is used by nodes to exchange messages, instead of using their communication tapes as they would normally do in the UC framework. That is, $\mathcal{F}_{\mathsf{link}}$ abstracts the AES link encryption, dummy messages, controlled traffic rates, and message re-ordering, and models the assumption of the impossibility to observe link messages for external adversaries. This assumption is broken down in Appendix B.7, by attempting to UC-realise the $\mathcal{F}_{\mathsf{link}}$ ideal functionality. A node X uses $\mathcal{F}_{\mathsf{link}}$ by giving it link-send subroutine inputs, with sid the session identifier of the protocol Π_{rtprop} of which node X is part, and Y the neighbor of X to which the message m must be sent. Upon such an input, $\mathcal{F}_{\mathsf{link}}$ simply gives a link-rcvd subroutine output to party Y, specifying the message m and the identity of X, letting Y know that the message is from its neighbor X (also, this models the fact that, in an Internet overlay, a node knows the IP address of its neighbors).

Functionality $\mathcal{F}_{\text{reg}}$ is responsible for delivering key pairs. It is necessary in order to then compose $\mathcal{F}_{\mathsf{rtprop}}$ with the full ideal functionality $\mathcal{F}$, and its role will thus be made clear in the modeling of the full protocol in Section 5.6. The functionality $\mathcal{F}_{\text{reg}}$ answers requests of the form (keys, sid, X), where sid is again the session identifier of Π_{rtprop}, and X specifies which party's keys are requested. The functionality maintains a table T of already generated keys, to answer consistently if the key pair of given party is requested more than once. Note that $\mathcal{F}_{\text{reg}}$ only answers to request for X's key pair if the request comes from the party X itself, or from the adversary if X is corrupted. $\mathcal{F}_{\text{reg}}$ also answers request from the ideal functionality $\mathcal{F}_{\mathsf{rtprop}}$: the latter needs these keys, in particular, to be able to output encryptions of one $c_{\mathtt{one}X \to R}$.

1 : **upon** input $(\mathsf{setup}, sid, src_X, dst_X)$: // Init. node
2 : Store src_X and dst_X ; Query $\mathcal{F}_{\text{reg}}$ and store (pk_X, sk_X)

3 : **upon** input $(\mathsf{proposer}, sid, (Y, cid), PS_{X\to R}, (Y', cid'))$: // Propose a route
4 : **if** $PS_{X\to R} = \mathsf{h}(dst_X{}^{src_X})$ **then** // Self-proposal
5 : Send $\langle \mathtt{rtprop} \| cid, c_1, c_2 \rangle$ to Y, where $c_1 \leftarrow \mathsf{Enc}(pk_X, dst_X)$ and $c_2 \leftarrow \mathsf{Enc}(pk_X, 1)$
6 : **else** // Relayed proposal
7 : Retrieve $(\mathsf{routing\text{-}table\text{-}entry}, PS_{X\to R}, c_{\mathsf{prop}X\to R}, c_{\mathsf{one}X\to R}, (Y', cid'))$
8 : Send $\langle \mathtt{rtprop} \| cid, c_1, c_2 \rangle$ to Y

9 : **upon** receiving message $\langle \mathtt{rtprop} \| cid, c_1, c_2 \rangle$ from Y: // Proposer ↔ Proposee
10 : **if** cid is unknown **then** // Proposee, 1st step
11 : Generate $(pk_X^{tmp}, sk_X^{tmp}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and send $\left\langle \mathtt{rtprop} \| cid, c'_1, c'_2 \right\rangle$ to Y,
12 : where $c'_1 \leftarrow \mathsf{ReEnc}_{\mathtt{nopk}}(c_2, \mathsf{ScExp}(c_1, src_X))$ and $c'_2 \leftarrow \mathsf{Enc}_{\mathtt{nopk}}(c_2, pk_X^{tmp})$
13 : Store $(\mathsf{proposee}, (Y, cid), c_{\mathsf{prop}} \coloneqq c_1, c_{\mathsf{one}} \coloneqq c_2, (pk_X^{tmp}, sk_X^{tmp}))$
14 : **elseif** cid relates to a proposal made by X **then** // Proposer, 2nd step
15 : Retrieve the information related to the ongoing route proposal with cid
16 : **if** the proposal is a self-proposal **then**
17 : Compute $PS_{X\to Y} = \mathsf{h}(\mathsf{Dec}(sk_X, c_1))$ and $pk_Y^{tmp} = \mathsf{Dec}(sk_X, c_2)$
18 : Send $\left\langle \mathtt{rtprop} \| cid, c'_1, c'_2 \right\rangle$ to Y, where $c'_1 \leftarrow \mathsf{Enc}(pk_Y^{tmp}, PS_{X\to Y})$ and $c'_2 \leftarrow \mathsf{Enc}(pk_Y^{tmp}, 1)$
19 : **else**
20 : Set $c_2 \leftarrow \mathsf{PlainMult}(c_2, pk^{tmp})$, and $c'_i \leftarrow \mathsf{ReEnc}_{\mathtt{nopk}}(c_{\mathsf{one}X\to R}, \mathsf{Dec}(sk_X, c_i))$ for $i \in \{1, 2\}$
21 : Sample $rcid \leftarrow\$ \{0,1\}^*$ and store $(\mathsf{relay}, (X, cid), \mathtt{null}, rcid, (Y', cid'))$
22 : Send $\left\langle \mathtt{rtproprelay} \| cid' \| rcid, c'_1, c'_2 \right\rangle$ to Y'
23 : **elseif** $\exists$ stored $(\mathsf{proposee}, (Y, cid), c_{\mathsf{prop}}, c_{\mathsf{one}}, (pk_X^{tmp}, sk_X^{tmp}))$ **then** // Proposee, 2nd step
24 : Get $PS_{X\to R} = \mathsf{Dec}(sk_X^{tmp}, c_1)$, and store $(\mathsf{routing\text{-}table\text{-}entry}, PS_{X\to R}, c_{\mathsf{prop}}, c_{\mathsf{one}}, (Y, cid))$
25 : Output $(\mathsf{proposee}, PS_{X\to R}, \mathsf{ReEnc}_{\mathsf{one}}(c_{\mathsf{one}}), (Y, cid))$

26 : **upon** receiving $\langle \mathtt{rtproprelay} \| cid \| rcid, c_1, c_2 \rangle$ from Y: // Return trip
27 : **if** $\exists \langle (Y, cid), \_, c_{\mathsf{one}X\to R}, NextHops \rangle \in RT$ **then** // Relay fwd
28 : **if** $NextHop = (Y', cid')$ and $rcid$ is unknown **then**
29 : Sample $rcid \leftarrow\$ \{0,1\}^*$, $(pk^{tmp}, sk^{tmp}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
30 : Store $(\mathsf{relay}, (Y, cid), rcid, sk^{tmp}, rcid', (Y', cid'))$
31 : Set $c_2 \leftarrow \mathsf{PlainMult}(c_2, pk^{tmp})$, and $c'_i \leftarrow \mathsf{ReEnc}_{\mathtt{nopk}}(c_{\mathsf{one}X\to R}, \mathsf{Dec}(sk_X, c_i))$ for $i \in \{1, 2\}$
32 : Send $\left\langle \mathtt{rtproprelay} \| cid' \| rcid', c'_1, c'_2 \right\rangle$ to Y'
33 : **elseif** $NextHop = \varnothing$ **then** // End-rcvr reached
34 : Compute $PS = \mathsf{h}(\mathsf{Dec}(sk_X, c_1))$, and get $pk^{tmp} = \mathsf{Dec}(sk_X, c_2)$
35 : Set $c'_1 \leftarrow \mathsf{Enc}(pk^{tmp}, PS)$ and $c'_2 \leftarrow \mathsf{Enc}(pk^{tmp}, 1)$
36 : Send $\left\langle \mathtt{rtproprelay} \| cid \| rcid, c'_1, c'_2 \right\rangle$ to Y
37 : **elseif** $\exists$ stored $(\mathsf{relay}, (Y', cid'), rcid', sk^{tmp}, rcid, (Y, cid))$ **then** // Relay back (or
38 : Set $c'_2 \leftarrow \mathsf{ReEnc}_{\mathsf{one}}(\mathsf{Dec}(sk^{tmp}, c_2))$, $c'_1 \leftarrow \mathsf{ReEnc}_{\mathtt{nopk}}(c'_2, \mathsf{Dec}(sk^{tmp}, c_1))$ // Proposer 2nd step)
39 : **if** $rcid' = \mathtt{null}$ **then** $h \leftarrow [\mathtt{rtprop} \| cid']$ **else** $h \leftarrow [\mathtt{rtproprelay} \| cid' \| rcid']$
40 : Send $\left\langle h, c'_1, c'_2 \right\rangle$ to Y'

Figure 5.5. – Description of Π_{rtprop} for Node X

Functionalities $\mathcal{F}_{\mathsf{link}}$ and $\mathcal{F}_{\text{reg}}$ being defined, Fig. 5.5 (page 108) describes Π_{rtprop}, the protocol for the topology dissemination phase. More specifically, it describes the pseudo-code of any given node X, defining all its actions during the topology dissemination phase. This consists in proposing routes, and handling route proposals from its neighbors. Since the code is from node X's point of view, it simultaneously depicts the behavior of X as proposee, proposer, end-receiver, or relay. With regards to the UC framework, the figure describes the code that each ITI runs in the real execution: there are as many running copies of this code as there are nodes in the network. It is assumed that every

node X maintains a local state during the protocol execution, and that, upon receiving an input from the environment or a message from another node, X behaves according to this state. Here, a node stores routing information (although, in the code, actual routing tables do not explicitly appear). In the code, the parties denoted Y or Y_i are the node's direct neighbors in the topology graph, while R denotes a distant receiver. Messages are denoted by $\langle msg \rangle$. The *entry points* of the code (marked by the **upon** keyword), either correspond to an input that the ITI of node X receives from the UC environment $\mathcal{E}$, or to the receiving of a message from another node (i.e. another ITI). We review these entry points in details in the following paragraphs. The first entry point (line 1 of the code) is a *setup* input, given once and only once by the environment to the ITI of node X. It instructs the node to initialise itself with specific src_X and dst_X values, and to obtain its keys from $\mathcal{F}_{\text{reg}}$. The second entry point (line 3) is a proposer input from the environment $\mathcal{E}$, asking the node to propose a route to its neighbor Y using a new circuit identifier cid. The specific route that X must propose is identified by the pseudonym $PS_{X \to R}$ of X towards the end-receiver R[4], along with the first hop (Y', cid'). $\mathcal{E}$ being semi-honest, this route is assumed present in the node's routing table (and more generally, all inputs are assumed well-formed). Node X may behave in two ways upon a proposer input, depending on whether X is asked to perform a self-proposal (line 4), or to relay another proposal (line 6). In any case, X eventually sends a `rtprop` message to the specified neighbor Y. This sending of message is implicitly done through the $\mathcal{F}_{\text{link}}$ ideal functionality: in order to simplify the code description, the instruction "send message m to Y" is used to formally mean "give subroutine input $(\text{link-send}, sid, Y, m)$". The same goes for the receiving of messages.

The third entry point (line 9), denotes the receiving of a `rtprop` message. As per the description of the protocol from Chapter 4, X can react in three different ways to this kind of messages, depending on whether it is the first (line 10), second (line 14), or third (line 23) message exchanged between the proposee and the proposer. Note that in the second case (when X is proposer w.r.t. this route proposal), two cases arise, depending on whether X is self-proposing or not. Also, in the last case, when X is a proposee receiving the final `rtprop` message, note that, X makes a proposer output to the environment, specifying the newly learned pseudonym, the encryption of one, and the next hop of the new route. The last entry point (line 26) deals with the receiving of a `rtproprelay` message, that must be relayed from the proposee to the receiver and back. This part of the code describes the behavior of node X as: relay on the way from proposee to the receiver (line 28), receiver (line 33), relay from receiver to proposee (line 37), and proposer (line 37 as well).

Then, Fig. 5.6 (page 110) models the ideal functionality $\mathcal{F}_{\text{rtprop}}$ corresponding to Π_{rtprop}. In $\mathcal{F}_{\text{rtprop}}$, and in the remainder of this chapter, the set of nodes in the network is denoted Ω, and the subset of honest (resp. corrupted) nodes is denoted Ω_h (resp. Ω_c). The code of the functionality $\mathcal{F}_{\text{rtprop}}$ is also written with the message-state paradigm. However, it is not given for a specific node X, but for the whole network. Indeed, in the ideal

[4]Note that the identity of R is not known to X. The term R is only here for denotational purposes.

1 : $\mathcal{F}_{\mathsf{rtprop}}$ internally runs an instance of Π_{rtprop} along with $\mathcal{F}_{\mathsf{link}}$.
2 : All inputs received by $\mathcal{F}_{\mathsf{rtprop}}$ are automatically passed on to the adequate party in Π_{rtprop}, and conversely for outputs (unless explicitly stated otherwise). Note that $\mathcal{F}_{\mathsf{rtprop}}$ knows all the routes and the relay nodes that compose them perfectly, as well as the *src* and *dst* values.

3 : **upon** input $(\mathsf{setup}, sid, src_Y, dst_Y)$ from dummy party Y:
4 : Pass the input to party Y in the internal Π_{rtprop} instance.

5 : **upon** input $(\mathsf{proposer}, sid, (X, cid), PS_{Y\to R}, (Z_1, cid_0))$ from dummy party Y:
6 : Pass the $\mathsf{proposer}$ input to party Y in Π_{rtprop}
7 : $\mathcal{F}_{\mathsf{rtprop}}$ deduces the end-receiver R, and the nodes $(Z_1, \ldots, Z_n)$ between Y and R.
8 : Let $Z_0 \coloneqq Y$, $Z_{n+1} \coloneqq R$
9 : Let $pred(Z_i, Z_j) = \mathbf{true}$ iff Z_i and Z_j are the 1st and last honest nodes in $(Z_0, \ldots, Z_{n+1})$
10 : Send the following to $\mathcal{S}\mathtt{im}$, *as the corresponding events happen in* Π_{rtprop}
11 : (I) **if** $Y, Z_1, \ldots, Z_n, R \in \Omega_c$ and $X \in \Omega_c \cup \Omega_h$ **then**
12 : $(\mathsf{rtprop}, sid, X \overset{cid}{\longleftrightarrow} Y \xrightarrow{cid_0} (Z_1 \overset{cid_1}{\longrightarrow} \ldots \overset{cid_{n-1}}{\longrightarrow} Z_n) \xrightarrow{cid_n} R, {dst_R}^{src_X})$ when R is solicited
13 : (II) **elseif** $Y, R \in \Omega_c$ and $X \in \Omega_c \cup \Omega_h$ and $\exists\ 1 \leq i \leq j \leq n$ s.t. $pred(Z_i, Z_j)$ **then**
14 : $(\mathsf{rtprop}, sid, ? \to (Z_j \overset{cid_{j+1}}{\longrightarrow} \ldots \overset{cid_n}{\longrightarrow} Z_n) \to R, {dst_R}^{src_X})$ when R is solicited
15 : $(\mathsf{rtprop}, sid, X \overset{cid}{\longleftrightarrow} Y \xrightarrow{cid_1} (Z_1 \overset{cid_1}{\longrightarrow} \ldots \overset{cid_{i-1}}{\longrightarrow} Z_i) \to ? \to R)$ when Y sends the last `rtprop` msg.
16 : (III) **elseif** $X, R \in \Omega_c$ and $\exists\ 0 \leq j \leq n$ s.t. $pred(Y, Z_j)$ **then**
17 : $(\mathsf{rtprop}, sid, X \overset{cid}{\longleftrightarrow} Y \to ?)$ when X receives the first `rtprop` message.
18 : $(\mathsf{rtprop}, sid, X \leftrightarrow ? \to (Z_j \overset{cid_{j+1}}{\longrightarrow} \ldots \overset{cid_n}{\longrightarrow} Z_n) \to R, {dst_R}^{src_X})$ when R is solicited.
19 : (IV) **elseif** $R \in \Omega_c$ and $X \in \Omega_h$ and $\exists\ 0 \leq j \leq n$ s.t. $pred(Y, Z_j)$ **then**
20 : $(\mathsf{rtprop}, sid, ? \to (Z_j \overset{cid_{j+1}}{\longrightarrow} \ldots \overset{cid_n}{\longrightarrow} Z_n) \to R, {dst_R}^{src_X})$ when R is solicited
21 : (V) **elseif** $X \in \Omega_c$ and $Y, R \in \Omega_h$ **then**
22 : $(\mathsf{rtprop}, sid, X \overset{cid}{\longleftrightarrow} Y \to ?)$ when X receives the first `rtprop` msg
23 : (VI) **elseif** $Y \in \Omega_c$ and $X \in \Omega_c \cup \Omega_h$ and $\exists\ 1 \leq i \leq n+1$ s.t. $pred(Z_i, R)$ **then**
24 : $(\mathsf{rtprop}, sid, X \overset{cid}{\longleftrightarrow} Y \xrightarrow{cid_1} (Z_1 \overset{cid_2}{\longrightarrow} \ldots \overset{cid_i}{\longrightarrow} Z_i) \to ?)$ when Y sends the last `rtprop` message

25 : For each sub-sequence $(Z_{i'}, \ldots, Z_{j'}) \subseteq (Z_1, \ldots, Z_n)$ of corrupted nodes *framed* by two honest nodes $Z_{i'}$ and $Z_{j'}$ that is solicited for relaying a message, send $(\mathsf{subpath}, sid, (Z_{i'} \overset{cid_{i'+1}}{\longrightarrow} \ldots \overset{cid_{j'}}{\longrightarrow} Z_{j'}))$, when the corrupted nodes in Π_{rtprop} are actually solicited (cid_k denotes both the *cid* and the *rcid* value of the k^{th} link)
26 : **if** $X \in \Omega_h$ and $R \in \Omega_c$ **then** wait for $(\mathsf{continue}, sid, X \overset{cid}{\longleftrightarrow} Y)$ from $\mathcal{S}\mathtt{im}$
27 : When X in Π_{rtprop} outputs $(\mathsf{proposee}, sid, PS_{X\to R}, c_{\mathsf{one}\,X\to R}, (Y, cid))$, relay it.

Figure 5.6. – The Ideal Functionality $\mathcal{F}_{\mathsf{rtprop}}$

execution, as per the UC framework, each node is represented by a *dummy ITI* (or dummy party) that accepts inputs from the environment and automatically pass them on to the ideal functionality. All the dummy parties use the same, unique instance of the ideal functionality. This explains why $\mathcal{F}_{\mathsf{rtprop}}$'s code entry points are inputs *from dummy parties* rather than the environment. Also, $\mathcal{F}_{\mathsf{rtprop}}$ only features entry points for inputs (for setup and $\mathsf{proposer}$ inputs, the same as Π_{rtprop}), and not for received message, since in a UC ideal execution, no messages are actually exchanged: all happens *within* the ideal functionality (this is the very principle of ideal functionalities in the UC framework). We further describe $\mathcal{F}_{\mathsf{rtprop}}$ in the following paragraphs.

Aside from the already mentioned high-level differences between $\mathcal{F}_{\mathsf{rtprop}}$ and Π_{rtprop}, note that the structure of their code is also quite different. While Π_{rtprop} is really a pseudo-code that could be turned in an actual implementation of the protocol, $\mathcal{F}_{\mathsf{rtprop}}$ is

a theoretical object that realises all the protocol within itself. This means in particular that it plays the role of all nodes. Actually, here, we have chosen to model $\mathcal{F}_{\mathsf{rtprop}}$ by making it run an internal instance of Π_{rtprop}, meaning that $\mathcal{F}_{\mathsf{rtprop}}$ re-creates the execution of the protocol internally: it executes copies of the code in Fig. 5.5 for each node, and make those nodes exchange messages *within itself*. When a route proposal concludes in the internal Π_{rtprop} instance, $\mathcal{F}_{\mathsf{rtprop}}$ accordingly makes a proposee output to $\mathcal{E}$ (*via* the adequate dummy party Y), by simply passing on the output of what X in the internally ran Π_{rtprop} outputs (see line 27 of the code). But $\mathcal{F}_{\mathsf{rtprop}}$ does not simply run Π_{rtprop}: as any UC ideal functionality, it also *explicitly* leaks information to the (ideal execution) adversary. That is, $\mathcal{F}_{\mathsf{rtprop}}$ explicitly sends information to the ideal adversary $\mathcal{S}\mathtt{im}$. These information leaks are actually most of the code, and all happen in the second entry point of the protocol, for proposer inputs (line 5).

Let us review in more details this part of the code. First, note that, to mirror the notations from Chapter 4, Y always denotes the proposer, X the proposee, R the end-receiver, and Z_i the relay nodes between proposer and end-receiver[5]. When the ideal functionality receives $(\mathsf{proposer}, sid, (X, cid), PS_{Y\to R}, (Y', cid'))$ from dummy party Y (meaning that the UC environment $\mathcal{E}$ instructed Y to propose a route), it first copies that input to Y in the internally ran instance of Π_{rtprop}. Then, $\mathcal{F}_{\mathsf{rtprop}}$ deduces all the information on the route to propose (line 7). Indeed, because $\mathcal{F}_{\mathsf{rtprop}}$ plays the role of all nodes in the network, it knows all *src* and *dst*, as well as all public and private keys. It also knows, at any point in time, the exact routing table of all nodes. Consequently, from the pseudonym $PS_{Y\to R}$ and the first hop (Z_1, cid_0), $\mathcal{F}_{\mathsf{rtprop}}$ deduces the full route, meaning all nodes, including the end-receiver. It is also assumed that $\mathcal{F}_{\mathsf{rtprop}}$ knows which nodes are corrupted and which ones are honest. From all these information, and depending on which nodes on the route are corrupted, $\mathcal{F}_{\mathsf{rtprop}}$ starts leaking information. More exactly, there are six different cases, depending on the corruption state of the proposee X, proposer Y, end-receiver R, and relay nodes Z_i. They are marked with roman numerals in Fig. 5.6. In addition to these six cases, the ideal functionality always leaks *intermediary corrupted sub-paths*, *i.e.* portions of routes made of corrupted relay nodes that are neither the proposee, proposer, or end-receiver (line 25).

To take an example, let us look at case (IV), on line 19 of the code. It corresponds to the case when the end-receiver is corrupted, possibly along with the last relay nodes $Z_{j+1}, \ldots,$ Z_n before R. The leaked information, here, is $(\mathsf{rtprop}, sid, ? \to (Z_j \overset{cid_{j+1}}{\to} \ldots \overset{cid_n}{\to} Z_n) \to R, {dst_R}^{src_X})$, giving the following information $\mathcal{S}\mathtt{im}$: the session identifier *sid*, the description of the last portion of the route (that is made of corrupted nodes from Z_{j+1} up to R), and the value ${dst_R}^{src_X}$. The question mark denotes the fact that the route *may* start before Z_j, but this portion of the route is not leaked. Note that this leaked information corresponds to what an actual adversary in an execution of our protocol over the Internet would learn about this particular route proposal in which nodes $Z_{j+1}, \ldots, Z_n$, and R are corrupted. These explicit leaks are one of the fundamental features of the UC framework: by looking at an ideal functionality, it is possible to see what the adversary learns, much

[5]Note that these naming naming conventions for parties does not appear in the code of Π_{rtprop}, because the latter is given from the point of view of one node, always denoted X.

easier so than by studying the full protocol.

Note that each information leak is made *at a specific moment*: for our proof that Π_{rtprop} UC-realises $\mathcal{F}_{\mathsf{rtprop}}$, it is necessary that the leaked information reflects the actual ordering of events in a real-world network. For instance, leak (IV) is made only when the corrupted end-receiver R is solicited in the Π_{rtprop} instance that $\mathcal{F}_{\mathsf{rtprop}}$ internally runs, since this would be the moment at which an actual network adversary would learn (in particular) the ${dst_R}^{src_X}$ value. Actually, this is the one of the main reasons why we set out $\mathcal{F}_{\mathsf{rtprop}}$ to internally run an instance of Π_{rtprop}, to be able to deduce a correct ordering of information leaks. Although this is not necessary *per se*, because this ordering could be explicitly enforced in the code of $\mathcal{F}_{\mathsf{rtprop}}$, this approach makes the code easier to write and read.

Finally, note that, in the UC framework, an ideal functionality is usually devoid of cryptographic material and operations. However, because this ideal functionality is going to be used as subroutine by the bigger protocol Π, which needs the encryptions of one acquired by route proposals (in particular to encrypt payload messages), there is no choice but to make $\mathcal{F}_{\mathsf{rtprop}}$ output it.

5.5.2. Π_{rtprop} UC-realises $\mathcal{F}_{\mathsf{rtprop}}$

Below is formulated the theorem defining the security of the route proposal part of the protocol. Because Π_{rtprop} uses $\mathcal{F}_{\mathsf{link}}$ and $\mathcal{F}_{\mathrm{reg}}$ as subroutines, it stated as standing *in the* $(\mathcal{F}_{\mathsf{link}}, \mathcal{F}_{\mathrm{reg}})$*-hybrid model*, in accordance with the UC framework terminology. This underlines the fact that those two ideal functionalities are indeed assumptions on the system model (in particular, $\mathcal{F}_{\mathsf{link}}$ is the assumption that network observer can not distinguish dummy messages from real ones). Figure 5.7 shows the relations between $\mathcal{A}$, $\mathcal{S}\mathtt{im}$, Π_{rtprop}, and $\mathcal{F}_{\mathsf{rtprop}}$ in the real and ideal executions. As it is standard in UC proofs [Can+02], $\mathcal{S}\mathtt{im}$ internally runs an copy of $\mathcal{A}$[6].

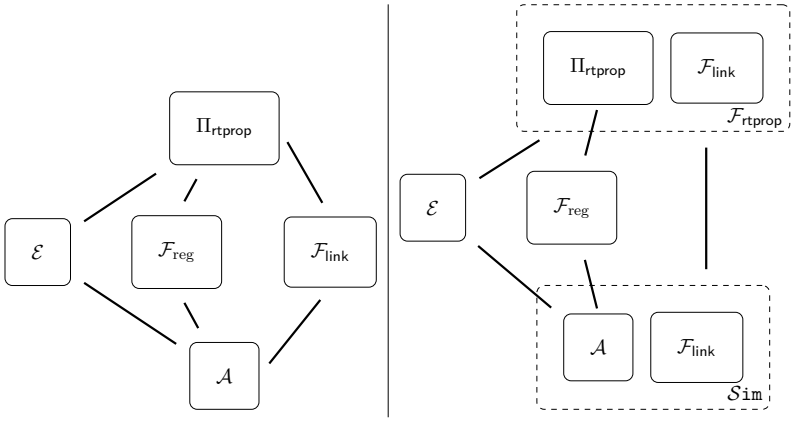

Figure 5.7. – Setup for Real (left) and Ideal (right) Executions

[6]More specifically, there are several standards for the proof setups, and $\mathcal{S}\mathtt{im}$ does not necessarily runs a copy of $\mathcal{A}$. See Appendix B.2 for a discussion on this matter.

Theorem 2 (Π_{rtprop} uc-realizes $\mathcal{F}_{\mathsf{rtprop}}$). *Assuming the IND-CPA, IK-CPA, and USS properties of the Elgamal scheme, the protocol Π_{rtprop} UC-realises $\mathcal{F}_{\mathsf{rtprop}}$ in the ($\mathcal{F}_{\mathrm{reg}}$, $\mathcal{F}_{\mathsf{link}}$)-hybrid model, in the presence of semi-honest static adversaries. That is, there exists $\mathcal{S}\mathtt{im}$ such that for all $\mathcal{A}$:*

$$\left\{\mathrm{EXEC}^{\mathrm{REAL}}_{\Pi_{\mathsf{rtprop}},\mathcal{A},\mathcal{E}}(z)\right\}_{\forall z\in\{0,1\}^*} \overset{c}{\equiv} \left\{\mathrm{EXEC}^{\mathrm{IDEAL}}_{\mathcal{F}_{\mathsf{rtprop}},\mathcal{S}\mathtt{im},\mathcal{E}}(z)\right\}_{\forall z\in\{0,1\}^*}$$

Proof Sketch 2 (Proof Sketch of Theorem 2). The full proof is given in Appendix B.3, where all the simulation cases are detailed (depending on the corruption state of the proposer, proposee, end-receiver, and relay nodes).

The system setup for the proof is as follows. $\mathcal{S}\mathtt{im}$ internally runs a copy of $\mathcal{A}$, relays the communications between $\mathcal{E}$ and $\mathcal{A}$ honestly. $\mathcal{S}\mathtt{im}$ also honestly relays $\mathcal{A}$'s queries to $\mathcal{F}_{\mathrm{reg}}$ for corrupted nodes' keys. Although $\mathcal{S}\mathtt{im}$ does see the key pairs of corrupted nodes in this process, it does not need this knowledge to perform the simulation: $\mathcal{F}_{\mathrm{reg}}$ is here only for convenience. Note that, the proof lies in the $\mathcal{F}_{\mathsf{link}}$-hybrid model. Thus, corrupted nodes (played by $\mathcal{A}$) and honest nodes (simulated by $\mathcal{S}\mathtt{im}$) exchange messages via *$\mathcal{F}_{\mathsf{link}}$ (which $\mathcal{S}\mathtt{im}$ internally runs as well). But more importantly, the $\mathcal{F}_{\mathsf{link}}$-hybrid model means that $\mathcal{S}\mathtt{im}$ does not need to simulate messages exchanged between honest nodes (since it is assumed that the adversary can not even observe them).*

$\mathcal{S}\mathtt{im}$ is not allowed to query $\mathcal{F}_{\mathrm{reg}}$ for the honest nodes' key. However, by the IK-CPA property (and because (public) keys are never revealed), $\mathcal{S}\mathtt{im}$ can safely replace them by random keys that it generates. The key pair of node X is denoted $(pk_{\mathcal{S}\mathtt{im}(X)}, sk_{\mathcal{S}\mathtt{im}(X)})$. Then, a crucial point behind the proof is that, because $\mathcal{S}\mathtt{im}$ gets to see all the inputs that $\mathcal{E}$ gives to corrupted nodes (as per the way the UC framework functions), $\mathcal{S}\mathtt{im}$ can perfectly know all the routes and links constructed between corrupted nodes. This knowledge allows $\mathcal{S}\mathtt{im}$ to infer information on the end-receivers of specific route proposals, and to be sure that when it sends a message with some cid to a specific corrupted node, this message will deterministically follow a known and expected route.

Then, the construction of $\mathcal{S}\mathtt{im}$ is mainly driven by the proposer *inputs from $\mathcal{A}$ to $\mathcal{E}$, the* proposee *outputs from $\mathcal{A}$ to $\mathcal{E}$, and the* rtprop *and* subpaths *leaks from $\mathcal{F}_{\mathsf{rtprop}}$. These inputs, outputs, and leaks contain all the necessary information to simulate honest nodes. Another crucial idea in the construction of the simulator is that every single* rtprop *or* subpath *leak can be simulated completely independently. Even, for instance, the two* rtprop *leaks in the (III) leaking case can be simulated independently,* i.e. *$\mathcal{S}\mathtt{im}$ can simulate them without knowing that they relate to the same route proposal. More generally, (almost) every portion of route made of corrupted nodes framed by two honest node can be simulated independently,* i.e. *$\mathcal{S}\mathtt{im}$ does not need to know that these sub-paths belong to the same route to correctly simulate them.*

As an example, the case (III) is simulated roughly as follows. The simulator $\mathcal{S}\mathtt{im}$ receives the first leak, ($\mathsf{rtprop}, sid, X \xleftrightarrow{cid_{X\text{-}Y}} Y \rightarrow ?$) when, in the internally ran instance of Π_{rtprop} within $\mathcal{F}_{\mathsf{rtprop}}$) the node X receives the first `rtprop` *message. From this leak, $\mathcal{S}\mathtt{im}$ knows that it must simulate the honest node Y making a route proposal, but does*

not however know whether it is self-proposing, and who the end-receiver is. Anyhow, $\mathcal{S}\texttt{im}$ *begins by simulating the sending of the* `rtprop` *message that* Y *would send to the corrupted node* X *in a route proposal: it sends*

$$\left\langle \texttt{rtprop} \| cid_{X\text{-}Y}, c_{\texttt{prop}} := \mathsf{Enc}(pk_{\mathcal{S}\texttt{im}(Y)}, r), c_{\texttt{one}} := \mathsf{Enc}(pk_{\mathcal{S}\texttt{im}(Y)}, 1) \right\rangle \tag{5.5}$$

At some point, the adversary $\mathcal{A}$*, on behalf of* X*, answers back (as it would do in a real execution) with ciphertexts* $\mathsf{Enc}(pk_{\mathcal{S}\texttt{im}(Y)}, r^{src_X})$ *and* $\mathsf{Enc}(pk_{\mathcal{S}\texttt{im}(Y)}, pk_X^{tmp})$*. The simulator* $\mathcal{S}\texttt{im}$ *can decrypt these ciphertexts, in particular to get* pk_X^{tmp}*. Then, the simulator waits for the dummy party* X *in* $\mathcal{F}_{\mathsf{rtprop}}$ *to output* $(\mathsf{proposee}, sid, PS_{X\to R}, c_{\texttt{one}X\to R}, (Y, cid_{X\text{-}Y}))$ *(this output is given by* $\mathcal{F}_{\mathsf{rtprop}}$ *directly to* $\mathcal{S}\texttt{im}$*, according to the way the UC framework works). At this point,* $\mathcal{S}\texttt{im}$ *can simulate the sending by* Y *of the final* `rtprop` *message, in particular by encrypting* $PS_{X\to R}$ *learned in the* $\mathsf{proposee}$ *outputs with* pk_X^{tmp}*.*

This terminates the simulation of the first leak of case (III) *. The second leak,* $(\mathsf{rtprop}, sid, X \leftrightarrow ? \to (Z_j \overset{cid_{j+1}}{\to} \ldots \overset{cid_n}{\to} Z_n) \to R, {dst_R}^{src_X})$*, is sent by* $\mathcal{F}_{\mathsf{rtprop}}$ *to* $\mathcal{S}\texttt{im}$ *when* R *in the internal* Π_{rtprop} *receives the* `rtproprelay` *message. This second leak, although it pertains to the same route proposal as the first leak, is simulated totally independently (*i.e. *no data from the simulation of the first leak is necessary here). Here,* $\mathcal{S}\texttt{im}$ *must simulate the honest node* Z_j *sending a* `rtproprelay` *to the corrupted node* Z_{j+1}*, and towards the corrupted end-receiver* R*. By construction of the simulation (which we do not detail here), the simulator* $\mathcal{S}\texttt{im}$ *is sure to have an encryption of one under the key* $pk_{Z_{j+1}} \cdot pk_{Z_{j+1}} \cdot \cdots \cdot pk_R$*, noted* $c_{\texttt{one}Z_j\to R}$*. Consequently,* $\mathcal{S}\texttt{im}$ *crafts ciphertexts* $\mathsf{Enc}_{\mathsf{nopk}}(c_{\texttt{one}Z_j\to R^*}, {dst_{R^*}}^{src_{X^*}})$ *and* $\mathsf{Enc}_{\mathsf{nopk}}(c_{\texttt{one}Z_j\to R^*}, pk^{tmp\prime})$*. Here,* ${dst_{R^*}}^{src_{X^*}}$ *comes from the leak, and* $pk^{tmp\prime}$ *is a fresh key generated by* $\mathcal{S}\texttt{im}$*. The simulator sends these ciphertexts in a* `rtproprelay` *message to* Z_{j+1}*, which is controlled by* $\mathcal{A}$*. Then, because* $\mathcal{A}$ *is semi-honest, and by the way routes are built in the simulation (as in a real execution), the message is ensured to reach* R*. However, since from* Z_{j+1} *to* R*, all nodes are corrupted, this is done by* $\mathcal{A}$ *herself (*$\mathcal{S}\texttt{im}$ *only needs to properly simulate the passing of messages through* $\mathcal{F}_{\mathsf{link}}$*). At some point, the adversary will send back, on behalf of* Z_{j+1}*, a* `rtproprelay` *message, meant to* Z_j*.* $\mathcal{S}\texttt{im}$ *receives that message on behalf of the honest node* Z_j*, and simply discards it. This concludes the second leak of case* (III) *.*

Constructing the simulator is only half of the proof. It remains to show that the simulation is indistinguishable from a real execution, from the point of view of $\mathcal{A}$ *and* $\mathcal{E}$*. First, note that, because the simulator is constructed to follow the order of leaks made by* $\mathcal{F}_{\mathsf{rtprop}}$ *(and of the inputs and outputs to and from* $\mathcal{E}$*), and because* $\mathcal{F}_{\mathsf{rtprop}}$ *itself is designed to make these leaks according to the ordering of events in a real execution (by running an internal instance of* Π_{rtprop}*),* $\mathcal{S}\texttt{im}$ *is sure to simulate all events in the adequate order for* $\mathcal{A}$*. Furthermore, cryptographically speaking, it can be formally shown that if there exists an adversary distinguishing between this simulation and a real execution with a non-negligible advantage, then it can be used to construct an adversary breaking the IND-CPA, IK-CPA or the USS property of the Elgamal PKE scheme with a non-negligible advantage. In particular, remark that, if an adversary is capable of distinguishing the simulation from a real execution based on the fact that the* `rtprop` *message (5.5) (shown in the above paragraph) encrypts a random group element* r *instead of* dst_R*, then it is*

possible to construct an adversary that breaks the IND-CPA property of the Elgamal scheme. Likewise, if the adversary can distinguish based on the fact that ciphertexts seen by the corrupted node X are completely independent from those seen by the corrupted node R, then it is possible to construct an adversary that breaks the USS property of the scheme (intuitively, this holds because in a real execution, nodes on the route re-encrypt ciphertexts).

5.5.3. Analysis of $\mathcal{F}_{\mathsf{rtprop}}$

Because the protocol is complex, in particular compared to small *cryptographic* protocols, the functionality $\mathcal{F}_{\mathsf{rtprop}}$, in itself does not directly make the properties of the route proposal mechanism appear. After some general remarks, this section studies the four desired properties of route proposals based on the leaks made by $\mathcal{F}_{\mathsf{rtprop}}$: route proposal homogeneity, route proposal indistinguishability, propagation untraceability, and return trip untraceability. Beforehand, however, the custom security definition used to do so is introduced.

5.5.3.a) General Remarks

It is clear, by inspection of $\mathcal{F}_{\mathsf{rtprop}}$, that if no node is corrupted, no information is given to the adversary (*i.e.* observers of the network learn nothing about route proposals and the created routes). As a result all four security properties of the route proposal mechanism hold perfectly against external adversaries.

In the presence of corrupted nodes, the adversary gets leaks mostly under the form of corrupted *sub-paths*, that are chains of corrupted nodes. Then, due to the way pseudonyms are computed in the protocol (as described by Fig. 4.5 in Chapter 4), during a route proposal with a corrupted end-receiver, the value ${dst_R}^{src_X}$ is leaked. Also, the first leaking case (I) is of particular interest, since it highlights an inherent limitation of the protocol: when the proposer Y, the end-receiver R and all relay nodes in between are corrupted, the adversary unavoidably learns the identity of the proposee X. Other leaking cases show, however, that if there is at least one honest node between the corrupted Y and the corrupted R, then the two halves of routes are leaked *independently*. As it appears clearly in the proof, if there is no concurrency among route proposals, $\mathcal{A}$ can trivially link these two halves together. However, in the general case, it is unclear what is the probability that the adversary makes this link: it depends at least on the traffic load (and the concurrency among route proposal), and the location in the topology graph of the nodes involved in each route proposal.

5.5.3.b) Formalisms to Define Route Proposal Security

For reasons described in the introductory section (Section 5.1.2), the properties of route proposals are proven (based on $\mathcal{F}_{\mathsf{rtprop}}$) using a custom security definition. This security definition is presented here. Recall that this definition should allow: (i) to define challenge in which the same intermediary corrupted sub-paths are solicited, (ii) to define challenges

on *portions* of route (rather *e.g.* than on a full return trip), and (ii) to issue challenges about one particular route proposal *out of the dynamics of the network.*

An other element to take into account when designing this security definition, is that $\mathcal{F}_{\mathsf{rtprop}}$ is not meant to be run by itself: it must be used in accordance to the route proposal policy. In particular, it is not possible to provide $\mathcal{F}_{\mathsf{rtprop}}$ with a proposer input for X, for a route that the latter does not known. Said otherwise, a proposer input can only be given to $\mathcal{F}_{\mathsf{rtprop}}$ if the latter made a proposee output for the corresponding route. Furthermore, it is not allowed to *e.g.* supply twice the same proposer input. This is a specificity of analysing a functionality like $\mathcal{F}_{\mathsf{rtprop}}$, that, in contrast with the analysis of a functionality that takes as input a send command instructing to start a communication session: while communication sessions can be arbitrarily launched by the adversary, route proposals must be made in a specific order. Consequently, we introduce a *wrapper* to the functionality $\mathcal{F}_{\mathsf{rtprop}}$, denoted W. The resulting functionality $W(\mathcal{F}_{\mathsf{rtprop}})$ takes input values of the following form:

$$I_W = (graph, RtPropPolicy, \{(\mathsf{setup}, sid, src_Z, dst_Z)\}_{Z \in \Omega})$$

That is, it takes as input a topology graph, a specific route proposal policy, and a setup input for each node. From that, W drives the topology dissemination phase, by iteratively producing all the proposer inputs for $\mathcal{F}_{\mathsf{rtprop}}$.

With this wrapper, we propose a custom security definitions based on *views*, where the view of the adversary in the execution of the functionality $W(\mathcal{F}_{\mathsf{rtprop}})(I_W)$ consists of all the leaks made by $\mathcal{F}_{\mathsf{rtprop}}$. More precisely, we consider that $W(\mathcal{F}_{\mathsf{rtprop}})(I_W)$ (for some input I_W) corresponds to: (i) the set of all proposer inputs that W provided to $\mathcal{F}_{\mathsf{rtprop}}$ for corrupted nodes, (ii) the set of all proposee outputs that $\mathcal{F}_{\mathsf{rtprop}}$ made for corrupted nodes, and (iii) the set of all information that is leaked by $\mathcal{F}_{\mathsf{rtprop}}$ during the execution. Then, to be able to define the security properties *without taking the dynamics of the network into account*, we denote $W(\mathcal{F}_{\mathsf{rtprop}})(I_W)[\mathtt{RP}(X{\leftrightarrow}Y{\rightarrow}R)]$ the information output by $W(\mathcal{F}_{\mathsf{rtprop}})(I_W)$, restricted to the information directly related to route proposal $\mathtt{RP}(X{\leftrightarrow}Y{\rightarrow}R)$.

With this formalism, the definition of *e.g.* route proposal indistinguishability roughly corresponds to the impossibility of distinguishing between $W(\mathcal{F}_{\mathsf{rtprop}})(I_W)[\mathtt{RP}(X{\leftrightarrow}Y{\rightarrow}R_0)]$ and $W(\mathcal{F}_{\mathsf{rtprop}})(I_W)[\mathtt{RP}(X{\leftrightarrow}Y{\rightarrow}R_1)]$, for honest end-receivers R_0 and R_1, and corrupted nodes X and Y. Here, $\mathtt{RP}(X{\leftrightarrow}Y{\rightarrow}R_0)$ and $\mathtt{RP}(X{\leftrightarrow}Y{\rightarrow}R_1)$ are called the *challenge route proposals.* However, this notion of security is trivial: without any context at all, given one of these two views, there is just no element whatsoever for the adversary to reason about the challenge. Therefore, we *manually* provide some context, namely by augmenting the views with encryptions of ones and pseudonyms. For instance, for route proposal indistinguishability, where X and Y are the corrupted nodes that are *challenged nodes* (since, in a real-world scenario, those are the node which must not distinguish between route proposals towards different end-receivers), the view $W(\mathcal{F}_{\mathsf{rtprop}})(I_W)[\mathtt{RP}(X{\leftrightarrow}Y{\rightarrow}R_b)]$ (for $b \in \{0,1\}$) is augmented with the pseudonym and encryption of one of X and Y towards end-receiver R_b. More generally, we define the adversary's view concerning route

proposal $\texttt{RP}(X{\leftrightarrow}Y{\rightarrow}R_b)$, for an input $i \in I_W$, and for challenged nodes in the set $\mathcal{N}$ as:

$$\begin{aligned}\mathsf{View}^{\mathcal{N}}_{\texttt{RP}(X\leftrightarrow Y\rightarrow R_b)}(i) \coloneqq \{o, \{Z, PS_{Z\rightarrow R}, c_{\texttt{one}Z\rightarrow R_b} \mid Z \in nodes(o) \cap \Omega_c\} \\ \mid o \in W(\mathcal{F}_{\mathsf{rtprop}})(i)[\texttt{RP}(X{\leftrightarrow}Y{\rightarrow}R_b)], \\ nodes(o) \cap (\Omega_c \setminus \mathcal{N}) = \emptyset\}\end{aligned}$$

That is, the view contains all outputs o of $W(\mathcal{F}_{\mathsf{rtprop}})(i)[\texttt{RP}(X{\leftrightarrow}Y{\rightarrow}R)]$ that contain the challenged nodes (and *only* those nodes), and for all the corrupted nodes Z appearing in these outputs, the view provides $PS_{Z\rightarrow R_b}$ and $c_{\texttt{one}Z\rightarrow R_b}$.

However, this is still not enough to yield a meaningful security definition. Actually, much more *context* can be provided to the adversary, while still being able to prove the properties. To better reflect a real-world scenario, where the adversary may have other corrupted nodes in the network than the challenged ones, we further introduce a set $\mathsf{Context}$ that specifies the pseudonyms and encryptions of one of other corrupted nodes in the network (according to a specific input $i \in I_W$). It is defined as follows, with $\mathcal{N}$ the set of challenged nodes, and R_0 and R_1 the two potential receivers:

$$\begin{aligned}\mathsf{Context}^{\mathcal{N}}_{R_0,R_1}(i) \coloneqq &\{(Z, src_Z, dst_Z, (pk_Z, sk_Z)) \mid Z \in \Omega_c\} \\ &\cup \{(Z, R', PS_{Z\rightarrow R'}, c_{\texttt{one}Z\rightarrow R'}, cid_{Z\rightarrow R'}) \\ &\qquad\qquad \mid Z \in \Omega_c, R' \in \Omega \setminus \{R_0, R_1\}\} \\ &\cup \{(Z, (R_a, PS_{Z\rightarrow R_a}, c_{\texttt{one}Z\rightarrow R_a}, cid_{Z\rightarrow R_a})) \\ &\qquad\qquad \mid Z \in \Omega_c \setminus \mathcal{N}, a \in \{0, 1\}\}\end{aligned}$$

This set contains, for all corrupted nodes $Z \in \Omega_c$, and for all end-receiver R' other than R_0 and R_1, the *de-anonymised* pseudonym and encryption of one of Z towards R'. It also contains, for all $Z \in \Omega_c$, and the *de-anonymised* pseudonym and encryption of one of Z towards R_0 and R_1. We say that those elements are *de-anonymised*, because they are explicitly accompanied with the identity of the end-receiver they are associated to. Adding the information in $\mathsf{Context}$ to the security definition aims at modeling a scenario where all corrupted nodes have de-anonymised all end-receivers, except the challenged nodes in $\mathcal{N}$ that did not de-anonymised R_0 and R_1. Indeed, in $\mathsf{Context}$, all corrupted nodes but the challenged ones get all the de-anonymised information about end-receivers, and in the View, the challenged nodes get the (anonymised) information for either R_0 or R_1 (plus all outputs from $W(\mathcal{F}_{\mathsf{rtprop}})$). This thus represents a very constraining scenario. Note however, that all the *context* we add to the adversarial views are of cryptographic nature. It does not put the challenge route proposals in the dynamic of the network, nor does it contain information on other route proposals than the challenge one (such as the intermediary corrupted sub-paths leaked during other route proposals). This effectively means that the proposed security definition, contrarily to the AnoA framework (in particular) does not take the dynamic of the network into account. It does, however try to *manually* account for the information the adversary may have obtained in past interactions, by providing *de-anonymised* pseudonyms and encryptions of one.

5.5.3.c) Definition and Proof of Route Proposal Security

With the proposed formalism, Definition 25 describes the four security properties of route proposals. In order to restrict the challenge scenario, we define each property on a subset $I_W^{\texttt{PROP}} \subseteq I_W$ of inputs. In particular, we restrict challenges to honest end-receivers, and to challenges that solicit the same intermediary corrupted sub-paths in both cases. For instance, route proposal indistinguishability is defined only over inputs $i \in I_W^{RPI}$ that yield executions in which the return trips of both challenge route proposals go through the exact same set of corrupted nodes. Our security results thus hold only for such executions, and security for route proposals soliciting different corrupted nodes in their return trip is not studied. Finally, in what follows, we sometimes denote corrupted sub-paths that make up a route as $\mathcal{Z}_k = \{Z_{k,1}, \ldots, Z_{k,n_k}\}$, for $k \in [1, K]$, where $K \in \mathbb{N}$, $n_k \leq l_{max}$ for all k. In each $\mathcal{Z}_k$, all nodes are corrupted, except one or both of the *end* nodes.

Definition 25 (Route Proposal Security). *The route proposal mechanism is said to be* secure *if* $\forall i \in I_W^{\texttt{PROP}}$,

$$\{view_0(i)\}_{i \in I_W} \overset{c}{\equiv} \{view_1(i)\}_{i \in I_W}$$

holds, with $view_b$ *and* $I_W^{\texttt{PROP}}$ *defined as follows for each of the four properties* $\texttt{PROP} \in \{RPH, RPI, PU, RTU\}$*:*

Route Proposal Homogeneity (RPH): *For* distinct *nodes* X, $R_0 = Y$, $R_1 = R$,

$$view_b(i) \coloneqq \Big(X, Y, R, \mathsf{Context}_{Y,R}^{\{X\}}(i), \mathsf{View}_{\texttt{RP}(X \leftrightarrow Y \rightarrow R_b)}^{\{X\}}(i)\Big)$$

Where I_W^{RPH} *is the set of input values for* $W(\mathcal{F}_{\mathsf{rtprop}})$ *that yield an execution for which* $X \in \Omega$*, and* $Y, R \in \Omega_h$*, and in which both* $\texttt{RP}(X \leftrightarrow Y \rightarrow Y)$ *and* $\texttt{RP}(X \leftrightarrow Y \rightarrow R)$ *occur, and the latter requires a return trip that* does not *solicit any corrupted node.*

Route Proposal Indistinguishability (RPI): *For* distinct *nodes* X, Y, R_0, R_1, *and any relay nodes in* $\mathcal{Z}_1, \ldots, \mathcal{Z}_K$*, let* $\mathcal{N} \coloneqq \{X, Y\} \cup \mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K$*. Define* $view_b(i)$ *as follows:*

$$view_b(i) \coloneqq \Big(X, Y, R_0, R_1, \mathsf{Context}_{R_0,R_1}^{\mathcal{N}}(i), \mathsf{View}_{\texttt{RP}(X \leftrightarrow Y \rightarrow R_b)}^{\mathcal{N}}(i)\Big)$$

Where I_W^{RPI} *is the set of input values for* $W(\mathcal{F}_{\mathsf{rtprop}})$ *that yield an execution for which* $X, Y \in \Omega$*,* $R_0, R_1 \in \Omega_h$*, all* Z_{k,i_k} *are corrupted except* Z_{1,n_k}*,* $Z_{k \neq 1,1}$ *and* $Z_{k \neq 1,n_k}$*, and where both* $\texttt{RP}(X \leftrightarrow Y \rightarrow R_0)$ *and* $\texttt{RP}(X \leftrightarrow Y \rightarrow R_1)$ *occur, and* both require a return trip soliciting the exact same corrupted sub-paths $\mathcal{Z}_1, \ldots, \mathcal{Z}_K$.

Propagation Untraceability (PU): *For* distinct *nodes* X, Y, X', Y', R_0, R_1, *and any relay nodes in* $\mathcal{Z}_1, \ldots, \mathcal{Z}_K$ *and* $\mathcal{Z}'_1, \ldots, \mathcal{Z}'_{K'}$*, let* $\mathcal{N} \coloneqq \{X, Y\} \cup \mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K$*,* $\mathcal{N}' \coloneqq \{X', Y'\} \cup \mathcal{Z}'_1 \cup \cdots \cup \mathcal{Z}'_{K'}$*. Define* $view_b(i)$ *as follows:*

$$view_b(i) \coloneqq \Big(X, X', Y, Y', R_0, R_1, \mathsf{Context}_{R_0,R_1}^{\mathcal{N} \cup \mathcal{N}'}(i), \mathsf{View}_{\texttt{RP}(X \leftrightarrow Y \rightarrow R_0)}^{\mathcal{N}}(i), \mathsf{View}_{\texttt{RP}(X' \leftrightarrow Y' \rightarrow R_b)}^{\mathcal{N}'}(i)\Big)$$

Where I_W^{PU} *is the set of input values for* $W(\mathcal{F}_{\mathsf{rtprop}})$ *that yields an execution in which* $X, Y, X', Y' \in \Omega$, $R_0, R_1 \in \Omega_h$, *all* Z_{k,i_k} *are corrupted except* Z_{1,n_k}, $Z_{k\neq 1,1}$ *and* $Z_{k\neq 1,n_k}$ *(and likewise for all nodes* $Z'_{k',i_{k'}}$*), and where* $\mathtt{RP}(X \leftrightarrow Y \rightarrow R_0)$, $\mathtt{RP}(X' \leftrightarrow Y' \rightarrow R_0)$ *and* $\mathtt{RP}(X' \leftrightarrow Y' \rightarrow R_1)$ *all occur, and the two latter ones require* a return trip soliciting the exact same intermediary corrupted sub-paths $\mathcal{Z}'_1, \ldots, \mathcal{Z}'_K$.

Return Trip Untraceability (RTU): *For nodes* R_0, R_1, *for any* distinct *nodes* X_0, Y_0, X_1, Y_1, *and any relay nodes in* $\mathcal{Z}_1, \ldots, \mathcal{Z}_K$ *and* $\mathcal{Z}'_1, \ldots, \mathcal{Z}'_{K'}$, *let* $\mathcal{N} \coloneqq \{X_0, Y_0\} \cup \mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K$, $\mathcal{N}' \coloneqq \{X_1, Y_1\} \cup \mathcal{Z}'_1 \cup \cdots \cup \mathcal{Z}'_{K'}$. *Define* $view_b(i)$ *as follows:*

$$view_b(i) \coloneqq \Big(X_0, X_1, Y_0, Y_1, R_0, R_1, \mathsf{Context}_{R_0,R_1}^{\mathcal{N}\cup\mathcal{N}'}(i), \mathsf{View}_{\mathtt{RP}(X_0 \leftrightarrow Y_0 \rightarrow R_0)}^{\mathcal{N}\setminus(\mathcal{Z}_{k_1}\cup\cdots\cup\mathcal{Z}_{k_2})}(i), \mathsf{View}_{\mathtt{RP}(X_b \leftrightarrow Y_b \rightarrow R_b)}^{\mathcal{Z}'_{k'_1}\cup\cdots\cup\mathcal{Z}'_{k'_2}}(i)\Big)$$

Where I_W^{RTU} *is the set of input values for* $W(\mathcal{F}_{\mathsf{rtprop}})$ *that yields an execution for which: (i)* $X_0, Y_0, X_1, Y_1 \in \Omega$, *and either* $R_0, R_1 \in \Omega_h$ *or* $R_0 = R_1 \in \Omega_c$*; (ii) all* Z_{k,i_k} *are corrupted except* Z_{1,n_k}, $Z_{k\neq 1,1}$ *and* $Z_{k\neq 1,n_k}$ *(and likewise for all nodes* $Z'_{k',i_{k'}}$*); (iii) both* $\mathtt{RP}(X_0 \leftrightarrow Y_0 \rightarrow R_0)$, $\mathtt{RP}(X_1 \leftrightarrow Y_1 \rightarrow R_0)$ *occur and respectively solicit intermediate sub-paths* $\mathcal{Z}_1, \ldots, \mathcal{Z}_K$ *and* $\mathcal{Z}'_1, \ldots, \mathcal{Z}'_{K'}$ *; and (iv) return trips of these two proposal have one (or more) intermediate sub-path(s) in common. That is,* $\exists k_1, k_2 \in [0, K-1]$, $\exists k'_1, k'_2 \in [0, K'-1]$ *s.t.* $(\mathcal{Z}_{k_1}, \mathcal{Z}_{k_1+1}, \ldots, \mathcal{Z}_{k_2}) = (\mathcal{Z}'_{k'_1}, \mathcal{Z}'_{k'_1+1}, \ldots, \mathcal{Z}'_{k'_2})$.

Note that route proposal homogeneity and indistinguishability are simply defined as the indistinguishability between two views. However, for propagation and return trip untraceability, the adversary gets a *commit view* (the same whether $b = 0$ or $b = 1$), and a *challenge view* (that depends on the value of b). Indeed, in propagation untraceability models the impossibility to link together two route proposal towards the same end-receiver. Thus, there must be a first view provided which *commits* to one end-receiver R_0, and a second view that either relates to a route proposal towards the same end-receiver R_0, or towards a different one R_1. The same reasoning applies to return trip untraceability. Secondly, note that, in the definition of each property, the challenge view is mainly focused on the end-receiver. This means that a mechanism secure by Definition 25 does not provide anonymity guarantees for proposers and proposees. That being said, in the context of the protocol, it is not a concern, since several (possibly many) proposees (that will later become *end-senders*) share the same route.

Theorem 3. *Assuming the indistinguishability of pseudonyms, and the IK-CPA property of the Elgamal scheme, the route proposal mechanism is secure w.r.t Definition 25. Namely, the adversarial advantage, for each property, is at most* $l_{max} \cdot \mathsf{Adv}^{\mathsf{ik\text{-}cpa}}(\lambda) + l_{max} \cdot \mathsf{Adv}^{\mathsf{ps-ind}}(\lambda)$, *for* l_{max} *the maximum route length.*

Proof Sketch 3 (Proof Sketch of Theorem 3). The full proof is given in Appendix B.4. By making the views of each property explicit (see the same Appendix), it appears clearly that, for each property, the only elements differing between $view_0(i)$ *and* $view_1(i)$ *are pseudonyms, encryptions of one, and cid values. By construction, these are not specified in the context (*i.e. *the end-receiver associated to them is not specified). The cid values are completely independent from the route (proposal) and from the end-receiver they designate. Hence, they do not give an advantage to the adversary. The advantage given by each pseudonym and encryption of one depend respectively on* $\mathsf{Adv}^{\mathsf{ps-ind}}(\lambda)$ *and* $\mathsf{Adv}^{\mathsf{ik\text{-}cpa}}(\lambda)$*, which are assumed negligible. In addition, the number of such pseudonyms and encryptions of one that differ is bounded by the number of relay nodes involved in the challenge, which is itself bounded by the maximum route length* l_{max}*. Therefore, through two hybrid game sequences, the theorem is easily proved, with the same methodology for all four properties.*

5.6. Security of the Protocol as a Whole

Building upon the results on the route proposal mechanism, this section studies the security of the protocol as a whole. It follows the same outline. The protocol's pseudo-code Π is first given. It is then modeled in an ideal functionality $\mathcal{F}$, and $\mathcal{F}$ is shown to UC-realise Π. Finally, $\mathcal{F}$ is analysed to prove the SA, RA, and SU properties with the AnoA framework, and MU with the custom security definition.

5.6.1. Modeling Π into an Ideal Functionality $\mathcal{F}$

The protocol Π is given in Fig. 5.9 (spanning over two pages, 122 and 123). It follows the same form as Π_{rtprop}: it is written in the message-state paradigm, from the point of view of one node X. We first describe the other ideal functionalities that Π uses as subroutines, and then comment the code itself.

The protocol Π uses the following ideal functionalities as subroutines: $\mathcal{F}_{\mathsf{link}}$, $\mathcal{F}_{\mathsf{offline}}$, $\mathcal{F}_{\mathsf{rtprop}}$, and $\mathcal{F}_{\mathrm{reg}}$. The former, $\mathcal{F}_{\mathsf{link}}$, is used in the exact same way as Π_{rtprop} (with implicit calls in the code whenever a message is sent/received). The second one, $\mathcal{F}_{\mathsf{offline}}$, is featured in Fig. 5.8. It models the *offline* exchange between an end-sender Alice and an end-receiver Bob, which is necessary prior to engaging in oriented communications. In essence, it is actually very similar to $\mathcal{F}_{\mathsf{link}}$, since it aims at modeling the fact that an external adversary can not observe the data exchanged during this interaction. Indeed, this exchange is supposed to happen outside of the network. The main difference with $\mathcal{F}_{\mathsf{link}}$ is however that, in $\mathcal{F}_{\mathsf{offline}}$, Alice stays anonymous even w.r.t Bob, since its identity is not output to Bob.

The third ideal functionality used by nodes in Π is $\mathcal{F}_{\mathsf{rtprop}}$. That is, instead of actually carrying out route proposals, nodes pass proposer subroutine inputs to $\mathcal{F}_{\mathsf{rtprop}}$ and get proposee subroutine outputs. Note that the same unique instance is used by all the ITIs of honest nodes, and by $\mathcal{A}$ as well. When it initialises (line 1 of Π's code), node X sends a setup input to $\mathcal{F}_{\mathsf{rtprop}}$, and then self-proposes, by giving one proposer input for each of its neighbors (line 5). Note that, by the way $\mathcal{F}_{\mathsf{rtprop}}$ is constructed, when X self-proposes to

1 : $\mathcal{F}_{\mathsf{offline}}$ models the offline exchange between Alice and Bob, prior to an oriented communication.
2 : **upon** input $(\mathsf{get}, sid, B, ocomid, c_{\mathsf{one}\,A\to I}, k)$ from party A:
3 : Output a copy of that input to B, and store $(sid, ocomid, A, B)$
4 : **upon** input $(\mathsf{got}, sid, ocomid, sh_1, c)$ from B:
5 : **if** $\exists$ a stored $(sid, ocomid, A, B)$ **then** Output a copy of this got input to A

Figure 5.8. – The Ideal Functionality $\mathcal{F}_{\mathsf{offline}}$

its neighbor Y_i by submitting a proposer input to $\mathcal{F}_{\mathsf{rtprop}}$, the latter automatically makes a proposee output to Y_i with the adequate pseudonym and encryption of one (see the entry point at line 6). Then, as already mentioned in previous sections, $\mathcal{F}_{\mathsf{rtprop}}$ does not include the route proposal policy: it does not include the decision to accept or refuse a route, and does not re-propose a route when one is learned. Therefore, in Π, when node X learns a new route (*i.e.* gets a proposee output from $\mathcal{F}_{\mathsf{rtprop}}$, at line 6), it runs *RtPropPolicy*, a function effectively abstracting the route proposal policy. This function returns two booleans, one stating whether X must accept the route, and the second stating if it should relay it. If X accepts a route, it creates an entry in its routing table RT. If it must relay the proposal, X gives an adequate proposer input to $\mathcal{F}_{\mathsf{rtprop}}$. Route proposals propagate in the network in this way. Note that corrupted nodes, controlled by $\mathcal{A}$, are assumed to behave in this way as well: $\mathcal{A}$, on behalf of a corrupted X, interacts with $\mathcal{F}_{\mathsf{rtprop}}$ to make route proposals.

Finally, Π makes uses $\mathcal{F}_{\mathrm{reg}}$. Similarly to Π_{rtprop} (and $\mathcal{F}_{\mathsf{rtprop}}$), nodes request their public keys to $\mathcal{F}_{\mathrm{reg}}$ (line 2). Here, the actual reason to be of $\mathcal{F}_{\mathrm{reg}}$ appears: its function is to ensure that the encryptions of one output by $\mathcal{F}_{\mathsf{rtprop}}$ are indeed usable by nodes in Π. Indeed, if the key pairs used internally by $\mathcal{F}_{\mathsf{rtprop}}$ did not match the keys used by nodes in Π, then the latter would encrypt payload messages with encryptions of one under (products of) public keys that they do not control, yielding payload messages impossible to decrypt. Note that generating keys in Π and passing them as input to $\mathcal{F}_{\mathsf{rtprop}}$ is not an option, since that would mean that, in the proof from Section 5.5.2 showing that Π_{rtprop} UC-realises $\mathcal{F}_{\mathsf{rtprop}}$, it is the UC environment that would have to provide the key pairs in input to the nodes. Since most of the security relies on the secrecy of these keys, the proof could not carry out. Secondly, note that $\mathcal{F}_{\mathrm{reg}}$ is set to only answers to the parties owning the key pair. More exactly, in the real execution, $\mathcal{F}_{\mathrm{reg}}$ answers only to honest node X from Π or to $\mathcal{F}_{\mathsf{rtprop}}$, and in an ideal execution it only answers to $\mathcal{F}$. In both executions, the adversary ($\mathcal{A}$ and $\mathcal{S}\mathtt{im}$ respectively) is allowed to query the keys of corrupted nodes (but not of honest ones). This *non-standard* modeling of key distributions seems unavoidable in order to split the protocol into Π_{rtprop} and Π, and to be able to propose a modular approach to the analysis of the full protocol. Its impact on the proof is however minimal, since it seems that a proof of the full protocol in one go (without dividing it into Π_{rtprop} and Π) would be possible *without* $\mathcal{F}_{\mathrm{reg}}$, by simply having nodes generate their key pairs locally.

1 : **upon** input $(\mathsf{setup}, sid, src_X, dst_X, neighbors)$: // Init node
2 : Query $\mathcal{F}_{\text{reg}}$ to get (pk_X, sk_X). Set $PS_{X\to X} \leftarrow \mathsf{h}(dst_X{}^{src_X})$
3 : Init routing table RT, and set entry $RT[0] = \langle \varnothing, PS_{X\to X}, \mathsf{Enc}(pk_X, 1), \mathtt{null}\rangle$
4 : Give subroutine input $(\mathsf{setup}, sid, src_X, dst_X)$ to $\mathcal{F}_{\mathsf{rtprop}}$. // Self-propose
5 : $\forall\, Y_i \in neighbors$: Sample cid_i, and give input $(\mathsf{proposer}, sid, (Y_i, cid_i), PS_{X\to X}, \varnothing)$ to $\mathcal{F}_{\mathsf{rtprop}}$

6 : **upon** output $(\mathsf{proposee}, sid, PS_{X\to R}, c_{\mathtt{one}X\to R}, (Y, cid))$ from $\mathcal{F}_{\mathsf{rtprop}}$: // Manage Rt. Props.
7 : Run $(accept, relay) \leftarrow RtPropPolicy(PS_{X\to R}, \dots)$
8 : **if** $accept = \mathbf{true}$ **then**
9 : Create RT entry $\langle \mathtt{null}, PS_{X\to R}, c_{\mathtt{one}X\to R}, (Y, cid)\rangle$
10 : **if** $relay \neq \emptyset$ **then** // Relay proposal
11 : $\forall\, Y_i \in reprop$: sample $cid_i \leftarrow\!\$\ \{0,1\}$ not yet in use with Y_i, and give subroutine input
12 : $(\mathsf{proposer}, sid, (Y_i, cid_i), PS_{X\to R}, (Y, cid))$ to $\mathcal{F}_{\mathsf{rtprop}}$.

13 : **upon** receiving message $m = \langle \mathtt{payload} \| cid, c_1, c_2\rangle$ from Y:
14 : **if** $\exists\, \langle (Y, cid), \_, c_{\mathtt{one}X\to R}, (Y', cid')\rangle \in RT$ **then** // Relay payload
15 : Set $c'_i \leftarrow \mathsf{ReEnc}_{\mathsf{nopk}}(c_{\mathtt{one}X\to R}, \mathsf{Dec}(sk_X, c_i))$ for $i \in \{1,2\}$
16 : Send $\left\langle \mathtt{payload} \| cid', c'_1, c'_2 \right\rangle$ to Y'
17 : **elseif** $\exists\, \langle (Y, cid), \_, \_, \mathtt{null}\rangle \in RT$ **then** // Receive payload
18 : Get $m_1 = \mathsf{Dec}(sk_X, c_1)$ and $m_2 = \mathsf{Dec}(sk_X, c_2)$
19 : **if** m_1 parses to $\mathtt{ocom} \| ocomid \| cnt$ **then** // I reached
20 : if $cnt \leq 5$, set $n = 1$, elseif $6 \leq cnt \leq 8$, set $n = 2$, else, set $n = 3$
21 : **goto** $OComInit(\mathsf{step} = I.n)$, and exit
22 : **elseif** m_1 parses to $\mathtt{ocom} \| ocomid \| cnt$ and $\exists$ stored $(ocomid, k)$ **then** // R reached
23 : $data$ by decrypting m_2 with k, and output $(\mathsf{rcvd}, sid, ocomid, data)$

24 : **upon** receiving message $m = \langle \mathtt{rtproprelay} \| cid \| rcid, c_1, c_2\rangle$ from Y:
25 : **if** $\exists\, \langle (Y, cid), \_, c_{\mathtt{one}X\to R}, NextHops\rangle \in RT$ **then** // Relay Alice $\to I$
26 : **if** $NextHop = (Y', cid')$ and $rcid$ is unknown **then** // Simple relay
27 : Sample $rcid \leftarrow\!\$\ \{0,1\}^*$, $(pk^{tmp}, sk^{tmp}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
28 : Store $(\mathsf{relay}, (Y, cid), rcid, sk^{tmp}, rcid', (Y', cid'))$
29 : Set $c_2 \leftarrow \mathsf{PlainMult}(c_2, pk^{tmp})$, and $c'_i \leftarrow \mathsf{ReEnc}_{\mathsf{nopk}}(c_{\mathtt{one}X\to R}, \mathsf{Dec}(sk_X, c_i))$ for $i \in \{1,2\}$
30 : Send $\left\langle \mathtt{rtproprelay} \| cid' \| rcid', c'_1, c'_2 \right\rangle$ to Y'
31 : **elseif** $NextHop = \varnothing$ **then** // I reached
32 : **goto** $OComInit(\mathsf{step} = I.1)$ and exit
33 : **elseif** $\exists$ stored $(\mathsf{relay}, (Y', cid'), rcid', sk^{tmp}, rcid, (Y, cid))$ **then** // Relay $I \to$ Alice
34 : **if** $rcid' \neq \mathtt{null}$ **then** // Simple relay
35 : Set $c'_2 \leftarrow \mathsf{ReEnc}_{\mathsf{one}}(\mathsf{Dec}(sk^{tmp}, c_2))$, $c'_1 \leftarrow \mathsf{ReEnc}_{\mathsf{nopk}}(c'_2, \mathsf{Dec}(sk^{tmp}, c_1))$
36 : Send $\left\langle \mathtt{rtproprelay} \| cid' \| rcid', c'_1, c'_2 \right\rangle$ to Y'
37 : **else** // Alice reached
38 : **goto** $OComInit(\mathsf{step} = Alice.2)$, and exit

Figure 5.9. – Description of Π for node X

With this description of the functionalities $\mathcal{F}_{\mathsf{link}}$, $\mathcal{F}_{\mathsf{offline}}$, $\mathcal{F}_{\text{reg}}$, and $\mathcal{F}_{\mathsf{rtprop}}$, a part of the code of Π have already been described. Although the rest of the code is constructed very similarly to the code of Π_{rtprop} in Fig. 5.5 (page 108), there are some major differences that are worth mentioning. Firstly, the routing table RT appears explicitly. Lookups in this table are often made directly in conditional statements. For instance, on line 14, the instruction "$\exists\, \langle (Y, cid), \_, c_{\mathtt{one}X\to R}, (Y', cid')\rangle \in RT$" tests the existence of a routing table entry with previous hop (Y, cid), and, if so, implicitly retrieves the encryption of one $c_{\mathtt{one}X\to R}$ and the next hop (Y', cid'). The underscore character simply means that the value of the field (here, the pseudonym) is not necessary in the code that follows. The second specificity of Π are the oriented communications themselves. In particular, the code entry point, corresponding to send inputs (on line 38, in the second part of the

38 : **upon** input $(\mathsf{send}, sid, ocomid, R, data)$: // Send $data$ to R
39 : **if** $\exists$ route initialised with $ocomid$ **then**
40 : **goto** $OcommInit(\mathsf{step} = Alice.3)$ and exit
41 : Pick a random RT entry $\langle \_, PS_{X\to I}, c_{\mathsf{one}X\to I}, NextHops\rangle$
42 : Give input $(\mathsf{get}, sid, R, ocomid, \mathsf{ReEnc}_{\mathsf{one}}(c_{\mathsf{one}X\to I}), k)$ to $\mathcal{F}_{\mathsf{offline}}$ with a key $k \leftarrow\$ \{0,1\}^*$
43 : Wait for subroutine output $(\mathsf{got}, sid, ocomid, sh_1, c)$ from $\mathcal{F}_{\mathsf{offline}}$, and **goto** $OComInit(\mathsf{step} = Alice.1)$

44 : **upon** subroutine output $(\mathsf{get}, sid, X, ocomid, c_{\mathsf{one}S\to I}, k)$ from $\mathcal{F}_{\mathsf{offline}}$: // Gen. shares of dst_R
45 : Store $(ocomid, k)$, and sample $sh_1 \leftarrow\$ \mathbb{G}$, and set $sh_2 = dst_X / sh_1$
46 : Set $c \leftarrow \mathsf{Enc}_{\mathsf{nopk}}(c_{\mathsf{one}S\to I}, sh_2)$
47 : Give subroutine input $(\mathsf{got}, sid, ocomid, sh_1, c)$ to $\mathcal{F}_{\mathsf{offline}}$

48 : $OComInit(\mathsf{step})$: // Steps of ocom. init.
49 : **if** $\mathsf{step} = Alice.1$ **then**
50 : Let sh_1 and $c = \mathsf{Enc}_{\mathsf{nopk}}(c_{\mathsf{one}X\to I}, sh_2)$ be the information received from $\mathcal{F}_{\mathsf{offline}}$
51 : Generate $(pk_X^{ocom}, sk_X^{ocom}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $(pk_X^{tmp}, sk_X^{tmp}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
52 : For $i \in [0,5]$, set $c_i \leftarrow \mathsf{Enc}_{\mathsf{nopk}}(c_{\mathsf{one}X\to I}, \mathtt{ocom}\|ocomid\|i)$, $c'_i \leftarrow \mathsf{Enc}_{\mathsf{nopk}}(c_{\mathsf{one}X\to I}, e[i])$
53 : with $e = (pk_X^{ocom}, sh_2, c_{sh_1}[0], c_{sh_1}[1], pk_X^{tmp}, pk_X^{tmp})$, and $(c_{sh_1}[0], c_{sh_1}[1]) \coloneqq \mathsf{Enc}(pk_X^{ocom}, sh_1)$
54 : Sample two reverse circuit id, $rcid_4$ and $rcid_5$, and $(Y, cid) \in NextHops$
55 : $\forall i \in [0,5]$, send $\left\langle h_i, c_i, c'_i \right\rangle$ to Y, with $h_i = \mathtt{payload}\|cid$ if $i \le 3$, else $h_i = \mathtt{rtproprelay}\|cid\|rcid_i$
56 : Store $(\mathsf{relay}, \varnothing, \mathtt{null}, sk^{tmp}, rcid, (Y, cid))$ along with $(ocomid, c_{\mathsf{one}X\to I})$.
57 : **if** $\mathsf{step} = I.1$ **then**
58 : Having received $c_0, c'_0, c_1, c'_1, c_2, c'_2, c_3, c'_3$ from $\mathtt{payload}$ messages.
59 : Having received c_4, c'_4, c_5, c'_5 from $\mathtt{rtproprelay}$ messages on link (Y, cid), and with $rcid_4$ and $rcid_5$
60 : Decrypt all these ciphertext, and recover $c_{sh_1} = \mathsf{Enc}(pk^{ocom}, sh_1)$, sh_2, pk^{ocom}, and pk^{tmp}
61 : Compute $c'' \leftarrow \mathsf{ReEnc}_{\mathsf{pk}}(pk^{ocom}, \mathsf{PlainMult}(\mathsf{ScExp}(c_{sh_1}, src_X), sh_2^{src_X}))$, and let $(c''[0], c''[1]) \coloneqq c''$
62 : Set $c_6 \leftarrow \mathsf{Enc}(pk^{tmp}, c''[0])$ and $c'_6 \leftarrow \mathsf{Enc}(pk^{tmp}, 1)$
63 : Set $c_7 \leftarrow \mathsf{Enc}(pk^{tmp}, c''[1])$ and $c'_7 \leftarrow \mathsf{Enc}(pk^{tmp}, 1)$
64 : For $i \in \{6,7\}$, send $\left\langle \mathtt{rtproprelay}\|cid\|rcid_{i-2}, c_i, c'_i \right\rangle$ to Y

65 : **if** $\mathsf{step} = Alice.2$ **then**
66 : Retrieve $(\mathsf{relay}, \varnothing, \mathtt{null}, sk^{tmp}, rcid, (Y, cid))$ along with $(ocomid, c_{\mathsf{one}X\to I})$
67 : Let $c_6 = \mathsf{Enc}(pk_X^{tmp}, c''[0])$ and $c_7 = \mathsf{Enc}(pk_X^{tmp}, c''[1])$ be the ciphertexts returned by I
68 : Reconstruct c'' by decrypting c_6 and c_7 with sk_X^{tmp}, and compute $PS_{I\to B} = \mathsf{h}(\mathsf{Dec}(sk_X^{ocom}, c''))$
69 : Set $c_8 \leftarrow \mathsf{Enc}_{\mathsf{nopk}}(c_{\mathsf{one}X\to I}, \mathtt{ocom}\|ocomid\|8)$ and $c'_8 \leftarrow \mathsf{Enc}_{\mathsf{nopk}}(c_{\mathsf{one}X\to I}, PS_{I\to B})$
70 : Send $\left\langle \mathtt{payload}\|cid, c_8, c'_8 \right\rangle$ to Y
71 : Store $(\mathsf{ocom\text{-}sender}, ocomid, c_{\mathsf{one}X\to I}, (Y, cid), cnt = 9, k)$ and **goto** $OComInit(\mathsf{step} = Alice.3)$
72 : **if** $\mathsf{step} = I.2$ **then**
73 : Having received c_8, c'_8 *via* $\mathtt{payload}$ message, decrypt and get $ocomid$ and $PS_{X\to R}$
74 : Retrieve RT entry $\langle \_, PS_{X\to R}, c_{\mathsf{one}X\to R}, NextHops'\rangle$ and sample $(Y'', cid'') \leftarrow\$ NextHops'$
75 : Store $(\mathsf{ocom\text{-}I}, ocomid, PS_{X\to R}, c_{\mathsf{one}R\to X}, (Y'', cid''))$

76 : **if** $\mathsf{step} = Alice.3$ **then** send $data$ as follows:
77 : Retrieve $(\mathsf{ocom\text{-}sender}, ocomid, c_{\mathsf{one}X\to I}, (Y, cid), cnt, k)$
78 : Set $c_{data} \leftarrow \mathsf{Enc}_{\mathsf{nopk}}(c_{\mathsf{one}X\to I}, \mathtt{ocom}\|ocomid\|cnt{+}{+})$ and $c'_{data} \leftarrow \mathsf{Enc}_{\mathsf{nopk}}(c_{\mathsf{one}X\to I}, \{data\}_k)$
79 : Send $\langle \mathtt{payload}\|cid, c_{data}, c_{data}\rangle$ to Y
80 : **if** $\mathsf{step} = I.3$ **then**
81 : Having received c_{data}, c'_{data}, *via* $\mathtt{payload}$ messages, decrypt and get $ocomid$, cnt, and $\{data\}_k$
82 : Retrieve $(\mathsf{ocom\text{-}I}, ocomid, PS_{X\to R}, c_{\mathsf{one}R\to X}, (Y'', cid''))$
83 : Set $c \leftarrow \mathsf{Enc}_{\mathsf{nopk}}(c_{\mathsf{one}X\to R}, \mathtt{ocom}\|ocomid\|cnt)$ and $c' \leftarrow \mathsf{Enc}_{\mathsf{nopk}}(c_{\mathsf{one}X\to R}, \{data\}_k)$
84 : Send $\langle \mathtt{payload}\|cid'', c, c'\rangle$ to Y''

Figure 5.9. – Description of Π for node X (Cont.)

code), instructs a node to send a message containing *data* to a specific end-receiver R using the oriented communication identifier *ocomid*. If this is the first such send input with this *ocomid*, X will initialise the session, by first picking an indirection node, and

requesting the *dst* shares to the specified receiver through $\mathcal{F}_{\mathsf{offline}}$ (line 42). But, because a node is allowed to send multiple messages in a single session, X may then receive other send inputs for this *ocomid*, with different *data* to send. In this case (line 39), X uses the already initialised session.

The third (and last) specificity of Π is the presence of a function *OComInit* (line 48), which is called *via* **goto** instructions (a non-standard notation that we introduce). Indeed, due to the complexity of the protocol, it is not possible to simply create one code entry point per *step* of the oriented communication initialisation procedure. In particular, the indirection node I typically receives a combination of `payload` and `rtproprelay` messages (corresponding respectively to the third and fourth code entry points), but that all contain material used in the *first step* of the oriented communication initialisation for the indirection node. Therefore, to group together all the operations relating to oriented communications and their initialisations, the code includes instructions such as "**goto** $OcomInit(\mathsf{step} = \mathsf{Alice.2})$", meaning that the code should jump to the *OComInit* function, to the second step of the oriented communication initialisation (for the role of the end-sender). We distinguish three steps each for the end-sender Alice and the indirection node I, that follow the description of oriented communication initialisation (see Fig 4.8 on page 85). Step one (lines 49-64) is the first round trip from Alice to I, step two (lines 65-75) is the computation and the communication form Alice to I of $PS_{I \to R}$, and the third step (lines 76-84) depicts the sending by Alice (and the relaying by I) of actual payload data.

We now turn to the description of the ideal functionality $\mathcal{F}$ corresponding to Π, detailed in Fig. 5.10 (page 125). Its form is quite similar to that of $\mathcal{F}_{\mathsf{rtprop}}$: it internally runs an instance of Π, and explicitly leaks information to the adversary. Similarly, $\mathcal{F}$ knows all routes, all pseudonyms, and all events in the network. There is one important element, however: $\mathcal{F}$ internally runs an instance of $\mathcal{F}_{\mathsf{rtprop}}$. Actually, $\mathcal{F}$ re-creates the full real execution inside itself. Again, this is a convenience that allows to leak the information to the adversary in the right order (see the discussion on this point in Section 5.5.1, on page 112).

The first part of $\mathcal{F}$'s code (lines 4-7) details how the topology dissemination is ran inside $\mathcal{F}$. Basically, the latter internally runs an instance of $\mathcal{F}_{\mathsf{rtprop}}$, and let nodes in its internal instance of Π (as well as the ideal execution adversary $\mathcal{S}\mathtt{im}$) communicate with it. Also, $\mathcal{F}$ relays all leaks from $\mathcal{F}_{\mathsf{rtprop}}$ to $\mathcal{S}\mathtt{im}$. Then, the second entry point of $\mathcal{F}$ (line 8) corresponds to the processing of send inputs (the only input type aside from setup). Here, similarly to $\mathcal{F}_{\mathsf{rtprop}}$, $\mathcal{F}$ leaks information on the oriented communication and its initialisation, depending on the corruption states of the end-sender S, the end-receiver R, and the indirection node I. However, $\mathcal{F}$ also lets $\mathcal{S}\mathtt{im}$ communicate choices that corrupted nodes have to make, in particular about the specific routes between S, I, and R, and *forces* these choices into the internally ran instance of Π. For instance, when S is corrupted, $\mathcal{F}$ sends sender-pick-route to $\mathcal{S}\mathtt{im}$ (line 12), which answers with a first hop of its choice (line 13). $\mathcal{F}$ then forces S in the internally ran Π instance to choose that first hop as well. This effectively models the influence of an actual network adversary on the protocol, that is, the adversary gets to chose the routes used by corrupted nodes.

$\mathcal{F}$ internally runs an instance of Π, along with $\mathcal{F}_{\mathsf{rtprop}}$, $\mathcal{F}_{\mathsf{link}}$ and $\mathcal{F}_{\mathsf{offline}}$.
upon input $(\mathsf{setup}, sid, src_X, dst_X, neighbors)$ from party X:
  Pass the input to the internal Π instance and ignore all subsequent setup inputs with sid of X.
  Relay all interactions between the internal instances of Π and $\mathcal{F}_{\mathsf{rtprop}}$ as follows: when X in Π wants to give subroutine input proposer to $\mathcal{F}_{\mathsf{rtprop}}$, pass it to the internal instance of $\mathcal{F}_{\mathsf{rtprop}}$, and *vice-versa* when $\mathcal{F}_{\mathsf{rtprop}}$ makes a proposee output for X.
  Relay all interaction between $\mathcal{S}\mathtt{im}$ and the internal instance of $\mathcal{F}_{\mathsf{rtprop}}$ as follows:
    – When the adversary $\mathcal{S}\mathtt{im}$ communicates a proposer input for a corrupted proposer Y, send it to $\mathcal{F}_{\mathsf{rtprop}}$ and manually update Y's routing table in the internal Π instance; and when $\mathcal{F}_{\mathsf{rtprop}}$ makes a proposee output for a corrupted proposee X, externally send it to the adversary, and analogously update X's routing table.
    – All leaks made by the internal $\mathcal{F}_{\mathsf{rtprop}}$ are immediately passed on to the adversary $\mathcal{S}\mathtt{im}$.

upon input $(\mathsf{send}, sid, ocomid, R, data)$ from party S:
  Pass the input to S in Π.
  if $ocomid$ designates a new ocom. session **then**
    if $S \in \Omega_c$ **then**
      Send $(\mathsf{sender\text{-}pick\text{-}route}, sid, ocomid)$ to $\mathcal{S}\mathtt{im}$
      Wait for $(\mathsf{sender\text{-}route}, sid, ocomid, (Y_S, cid_S))$ from $\mathcal{S}\mathtt{im}$
      Force the choice of S in Π on that route, effectively also forcing the choice of indirection node I
    else
      Let S in Π choose a routing table entry, defining the identity of I and the first hop (Y_S, cid_S).
    if $R \in \Omega_c$ **then**
      Send $(\mathsf{offline}, sid, ocomid, R)$ to $\mathcal{S}\mathtt{im}$
    if $I \in \Omega_c$ **then**
      Send $(\mathsf{I\text{-}pick\text{-}route}, sid, ocomid)$ to $\mathcal{S}\mathtt{im}$
      Wait for $(\mathsf{I\text{-}route}, sid, ocomid, (Y_I, cid_I))$ from $\mathcal{S}\mathtt{im}$
      Force the choice of I in Π on that route
      Then, for each message that I receives from S in Π:
        Send $(\mathsf{ocom\text{-}I}, sid, (Z_j \overset{cid_{j+1}}{\rightarrow} \ldots \overset{cid_n}{\rightarrow} I), \mathtt{flag}, ocomid \| cnt)$ to $\mathcal{S}\mathtt{im}$,
          where Z_j is the last honest node between S and I
        In addition, send $PS_{I \rightarrow R}$, when I receives the pseudonym
    if $S \in \Omega_c$ **then** when S receives the two rtproprelay messages from I in Π
        Send $(\mathsf{ocom\text{-}sender}, sid, (Z_i \overset{cid_{i-i}}{\rightarrow} \ldots \overset{cid_1}{\rightarrow} S), ocomid \| cnt, {dst_R}^{src_I})$ to $\mathcal{S}\mathtt{im}$
          where Z_i is the first honest node between S and I
  Let $I, (Y_S, cid_S), (Y_I, cid_I), cnt$ be the indirection node, the end-sender's first hops, the indirection node's first hop, and the counter associated to $ocomid$
  if $I \in \Omega_c$ **then** when I in Π receives the payload message with $data$
    Send $(\mathsf{ocom\text{-}I}, sid, (Z_j \overset{cid_{j+1}}{\rightarrow} \ldots \overset{cid_n}{\rightarrow} I), ocomid \| cnt)$ to $\mathcal{S}\mathtt{im}$
    if $R \in \Omega_c$ as well **then** also send $data$
  if $R \in \Omega_c$ **then** when R in Π receives the payload message with $data$
    Send $(\mathsf{ocom\text{-}rcvr}, sid, (Z_j \overset{cid_{j+1}}{\rightarrow} \ldots \overset{cid_n}{\rightarrow} R), ocomid \| cnt, data)$ to $\mathcal{S}\mathtt{im}$
  For each sub-sequence $(Z_{i'}, \ldots, Z_{j'}) \subseteq (Z_1, \ldots, Z_n)$ of corrupted nodes *framed* by two honest nodes $Z_{i'}$ and $Z_{j'}$, send $(\mathsf{subpath}, sid, (Z_{i'} \overset{cid_{i'+1}}{\rightarrow} \ldots \overset{cid_{j'}}{\rightarrow} Z_{j'}), \mathtt{flag})$, to $\mathcal{S}\mathtt{im}$ when the corrupted nodes in Π are actually solicited (here, cid denotes both the cid and the $rcid$ value of each link)
  if $R \in \Omega_h$ and $\{S, I\} \cap \Omega_c \neq \emptyset$ **then**
    Send $(\mathsf{continue?}, sid, ocomid \| cnt)$ to $\mathcal{S}\mathtt{im}$
    Wait for $(\mathsf{continue}, sid, ocomid \| cnt)$ from $\mathcal{S}\mathtt{im}$
  Output $(\mathsf{rcvd}, sid, ocomid, data)$ to party R.

Figure 5.10. – The Ideal Functionality $\mathcal{F}$

This design is also *necessary* for the proof that Π UC-realises $\mathcal{F}$ to carry out. Other than these *choices* that $\mathcal{S}\mathtt{im}$ is allowed to input into $\mathcal{F}$, the latter merely leaks information

about *corrupted sub-paths*, very similarly to $\mathcal{F}_{\mathsf{rtprop}}$. In particular, it leaks *intermediary corrupted sub-paths*, and, when S, I, and/or R are corrupted, it also leaks the end portion of route that is corrupted, every time one of those receives a message. When relevant, the leak is accompanied with the (decrypted) contents of these messages. Finally, because several *data* payloads can be sent in the same oriented communication session, several send inputs may be given with the same *ocomid*. $\mathcal{F}$ takes care to "run" the oriented communication initialisation only once, when a send input specifies a new *ocomid* (this is the reason to be of the conditional on line 10).

5.6.2. Π UC-Realizes $\mathcal{F}$

Figure 5.11 shows the setup for the proof, and the relations between all parties. Next, the theorem on the UC-realisation of $\mathcal{F}$ is formulated. Note that it is proven under the assumption that end-senders and indirection nodes do not collude, and in particular, that a corrupted end-sender never chooses a corrupted indirection node.

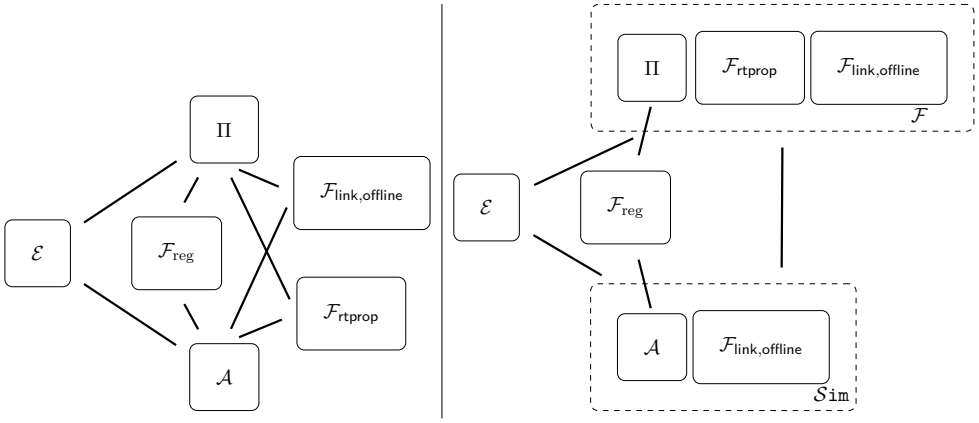

Figure 5.11. – Setup for Real (left) and Simulated (right) Executions

Theorem 4 (Π UC-Realizes $\mathcal{F}$). *Assuming the IND-CPA, IK-CPA, and USS properties of the Elgamal scheme, and assuming that end-senders and indirection nodes do not collude, the protocol Π UC-realises the ideal functionality $\mathcal{F}$ in the $(\mathcal{F}_{\mathrm{reg}}, \mathcal{F}_{\mathsf{link}}, \mathcal{F}_{\mathsf{offline}}, \mathcal{F}_{\mathsf{rtprop}})$-hybrid model in the presence of semi-honest static adversaries. That is, there exists $\mathcal{S}\mathtt{im}$ such that for all $\mathcal{A}$:*

$$\left\{\mathrm{EXEC}^{\mathrm{REAL}}_{\Pi,\mathcal{A},\mathcal{E}}(z)\right\}_{\forall z\in\{0,1\}^*} \overset{c}{\equiv} \left\{\mathrm{EXEC}^{\mathrm{IDEAL}}_{\mathcal{F},\mathcal{S}\mathtt{im},\mathcal{E}}(z)\right\}_{\forall z\in\{0,1\}^*}$$

Proof Sketch 4 (Proof Sketch of Theorem 4). The full proof is given in Appendix B.5, where all the simulation cases are detailed (depending on the corruption state of the end-sender, indirection node, end-receiver, and relay nodes). The system setup for the proof is roughly the same as for the proof of Theorem 2, to the difference that $\mathcal{F}$ internally runs a instance of $\mathcal{F}_{\mathsf{rtprop}}$.

At the beginning of the simulation, the topology dissemination phase is driven by $\mathcal{F}$ for honest nodes (or more exactly, by the Π instance in $\mathcal{F}$), and by $\mathcal{A}$ for corrupted ones. That is, the actual route proposals are performed within the internal $\mathcal{F}_{\mathsf{rtprop}}$ instance of $\mathcal{F}$, but the route proposal policy (the decision to accept/refuse routes, and to relay them) is run by $\mathcal{F}$ and $\mathcal{A}$. $\mathcal{S}\mathtt{im}$ relays proposer *inputs and* proposee *outputs between $\mathcal{A}$ and $\mathcal{F}$. By that, $\mathcal{S}\mathtt{im}$ learns all the portions of routes that are made of corrupted nodes (as in proof of Theorem 2). Also, to be able to simulate the oriented communications, $\mathcal{S}\mathtt{im}$ modifies the* Cone *encryptions in* proposee *outputs to replace them with encryptions under keys it controls,* i.e. *so that any honest node at the end of a sub-path has the ability to decrypt ciphertexts in the message they receive (as in proof of Theorem 2). This modification is indistinguishable, by the USS and IK-CPA properties.*

With this setup, the rest of the simulation uses exactly the same methods as in the proof of Theorem 2, and the proof holds by similar arguments. Messages are crafted and delivered *to corrupted end of routes (here, end-senders, indirection nodes, and end-receivers), using the last honest node on the route. Again, $\mathcal{F}$ leaks all the necessary information to do so.*

There are a few differences, however. Firstly, although intermediary corrupted sub-paths are simulated independently from each other, end of routes are not*. Indeed, the ocomid value binds all the messages in the same oriented communication. This allows $\mathcal{S}\mathtt{im}$ to correctly deliver the information proper to the oriented communications (such as the values sh_2, pk^{ocom}, or pk^{tmp}). In particular, when both the indirection node and the end-receiver are corrupted, care is taken to deliver them the same encrypted payloads $\{data\}_k$. The other difference is that $\mathcal{S}\mathtt{im}$ runs a simulated version of $\mathcal{F}_{\mathsf{offline}}$, and by that, gets to observe all offline interactions involving a corrupted end-sender or end-receiver. On the other hand, $\mathcal{S}\mathtt{im}$ must also answer on behalf of honest end-receivers contacted by corrupted end-senders though $\mathcal{F}_{\mathsf{offline}}$, and in particular generate shares sh_1 and sh_2 without actually knowing dst_R. However, by the security of the secret sharing mechanism, $\mathcal{S}\mathtt{im}$ can simply generate random shares, since end-senders and indirection nodes are assumed not to collude. Lastly, in order to synchronize the internal states of the simulation and of the ideal functionality, the latter lets $\mathcal{S}\mathtt{im}$ communicate the non-deterministic decisions made by $\mathcal{A}$, such as the indirection node chosen by a corrupted end-sender. This ensures that, when an intermediary sub-path are leaked, it corresponds to an actual sub-path that needs to be simulated. $\mathcal{S}\mathtt{im}$ can thus blindly simulate all intermediary corrupted sub-paths as they are leaked.*

Finally, the indistinguishability between the simulated and real executions stems from the IND-CPA, IK-CPA, and USS properties, similarly to proof of Theorem 2.

5.6.3. Analysis of $\mathcal{F}$

Based on the ideal functionality $\mathcal{F}$, this section aims at studying the SA, RA, SU and MU properties. Indeed, as it is the case with $\mathcal{F}_{\mathsf{rtprop}}$, the ideal functionality being quite complex, the security properties of the protocol do not appear trivially.

5.6.3.a) General Remarks

Just as $\mathcal{F}_{\mathsf{rtprop}}$, the functionality $\mathcal{F}$ does not leak any information to network observers, in particular since we are in the $\mathcal{F}_{\mathsf{link}}$-hybrid model (in which external observers are assumed not to be able to observe messages exchanged between neighboring nodes). Thus, SA, RA, SU, and MU hold perfectly against external adversaries. Then, in the presence of corrupted nodes, the nature of information leaked by $\mathcal{F}$ is similar to that of $\mathcal{F}_{\mathsf{rtprop}}$: it mostly consists of *sub-paths*. Again, because of the way $PS_{I \to R}$ is computed during an oriented communication initialisation (as defined in Fig. 4.8 in Chapter 4), when the end-sender S is corrupted, $\mathcal{F}$ leaks the value ${dst_R}^{src_I}$. Interestingly, $\mathcal{F}$ *must* allow the adversary to *input* $\mathcal{F}$ with own random choices (such as the route used by an end-sender or indirection node), or else the proof does not follow through. More accurately, in our approach to construct the simulator, in order to be able to prove that the ideal and real executions are indistinguishable, inputting the adversary's random choice into $\mathcal{F}$ is necessary. For instance, if a route from the end-sender to the indirection node is not the same in the simulation and in the ideal functionality, the latter will leak intermediary sub-paths that do not correspond to the ones that the adversary expects to observe (thus possibly allowing the adversary to distinguish the simulation from a real execution).

Then, from the (full) proof, it is clear that intermediary corrupted sub-paths can be simulated independently, which means that in general, corrupted relay nodes (other than indirection nodes) can not link all the messages belonging to the same session with probability one (although all messages in a session take the same route from S to I and then I to R). This also means that a given relay node can not distinguish between a message on a first leg S-I where I is an indirection node, and a message on a second leg I'-I where I is an end-receiver. However, this property has limited interest in the formal analysis. Indeed, the frequency at which these cases arise depend on the traffic load and the number of concurrent communications. yet, in the AnoA framework's approach, a *worst-case* scenario is assumed, and the traffic load is actually *controlled* by the adversary (this amounts to assuming that there is only one communication at a time in the network, and no concurrency among communications).

In accordance with the considered approach described in Section 5.1.2, we set out to analyse $\mathcal{F}$ with the AnoA framework, in order to prove the SA, RA, and SU properties. In a second time, the MU property is studied with the same formalism as the route proposal properties.

5.6.3.b) Proving SA, RA, and SU with AnoA

In this section, the SA, RA, and SU properties are proved following the same methodology that Backes *et al.* apply to Tor [Bac+13]. The *adjacency functions* α_{SA}, α_{RA}, and α_{SU}, which formally define each property in the AnoA framework, are given in Fig 5.12. More accurately, they are formulated with the *adaptive AnoA* setup, but for $n = 1$ challenge session only (see [Bac+13, Section 5] for detail on the adaptive AnoA setup). Recall that adjacency functions are sort of *hooks* into the challenger, the latter being fixed by the AnoA framework (*i.e.* its *behavior* is fixed). Basically, the adversary produces

(polynomially) as many $r = (S, R, data, ocomid)$ input values as she wants for arbitrary end-senders and end-receivers. She gives those inputs to the challenger, who passes them as send inputs to $\mathcal{F}$. At some point, $\mathcal{A}$ produces a *challenge action* modeled by r_0 and r_1, representing two different oriented communication session *e.g.* with two different end-senders for SA). The adjacency function is applied to (r_0, r_1, b), which, for simple properties, returns r_b after basic input checks, and the challenger runs the protocol on it. However, the adjacency function is allowed to perform further processing and return a mix of r_0 and r_1, if it is pertinent. As usual, the objective of $\mathcal{A}$ is to guess the value of the bit b (knowing the structure of the adjacency function considered).

$\alpha_{SA}(st, r_0 = (S_0, R, data, ocomid), r_1 = (S_1, \_, \_, \_), b)$:
  if ($st = fresh$ or $st = (S_0, S_1, R, ocomid)$) and $S_0, S_1 \in \Omega_h$ and topology dissemination is over **then**
    Output $((S_b, R, data, ocomid), st := (S_0, S_1, R, ocomid))$

$\alpha_{RA}(st, r_0 = (S, R_0, data, ocomid), r_1 = (\_, R_1, \_, \_), b)$:
  if ($st = fresh$ or $st = (R_0, R_1, S, ocomid)$) and $S, R_0, R_1 \in \Omega_h$ and topology dissemination is over **then**
    Output $((S, R_b, data, ocomid), st := (R_0, R_1, S, ocomid))$

$\alpha_{SU}(st, r_0 = (S_0, R_0, data, ocomid), r_1 = (S_1, R_1, \_, ocomid'), b)$:
  if $S_0, S_1, R_0, R_1 \in \Omega_h$ and topology dissemination is over **then**
    if $st = fresh$ **then**
      $a \leftarrow_{\$} \{0, 1\}$
      Output $((S_a, R_a, data, ocomid), st := (S_0, R_0, S_1, R_1, a, ocomid, stage1))$
    elseif $st = (S_0, R_0, S_1, R_1, a, ocomid, stage1)$ and $ocomid = ocomid'$ **then**
      Output $((S_a, R_a, data, ocomid), st)$
    elseif $st = (S_0, R_0, S_1, R_1, a, ocomid, stage1)$ and $ocomid \neq ocomid'$ **then**
      if $b = 0$ **then** $a' := a$ **else** $a' := 1 - a$
      Output $((S_{a'}, R_{a'}, data, ocomid'), st := (S_0, R_0, S_1, R_1, a', ocomid', stage2))$
    elseif $st = (S_0, R_0, S_1, R_1, a', ocomid', stage2)$ **then**
      Output $((S_{a'}, R_{a'}, data, ocomid'), st)$

Figure 5.12. – AnoA Adjacency Functions for SA, RA, and SU

The functions α_{SA} and α_{RA} described in Fig 5.12 take as input two tuples each specifying an end-sender, an end-receiver, some *data* to send, and an oriented communication identifier. In α_{SA}, for instance, the first tuple is $r_0 = (S_0, R, data, ocomid)$, and the second tuple is $r_1 = (S_1, \_, \_, \_)$. The underscore characters mean that α_{SA} expects that r_1 specifies the same R, *data*, and *ocomid* as r_0. Said otherwise, r_0 and r_1 are expected to differ only in the end-sender they specify, *i.e.* S_0 or S_1. The other inputs of α_{SA} are st, the *state* of the challenge, and the bit b. The latter is fixed by the challenger,

which itself receives it in input as well (see Section 5.1.1.b on the AnoA framework). The state st is used to manage the challenge, for instance to ensure that the challenge is made on only one session [Bac+13]. The code of α_{SA} merely consists in performing a few verifications, and outputting r_b and an updated state st. These verifications define (or, more accurately, restrict) the challenge scenario: if the verifications fail, α_{SA} does not output anything, meaning that the r_0 and r_1 input values are invalid for the challenge. Here, we verify: (i) that the state is equal $fresh$ (its default value from the challenger) or equal to $(S_0, S_1, R, ocomid)$, in order to ensure that the challenge focuses on only one communication session, from S_b to R with $ocomid$ (however, several messages can be sent in that session, *i.e.* α_{SA} may be called several times with the same S_b, R, and $ocomid$); (ii) that S_0 and S_1 are both honest, since it otherwise does not make sense to test sender anonymity; and (iii) that the topology has been fully disseminated, in order to avoid cases where one or both of the end-senders are unable to contact the specified end-receiver R (*e.g.* because no routes at all have been proposed at the time of the challenge). If all these conditions are met, α_{SA} outputs r_b (and updates st), and the challenger consequently provides input $(\mathsf{send}, sid, ocomid, R, data)$ for S to $\mathcal{F}$. The latter thus perform leaks, that the challenger forwards to $\mathcal{A}$, who must then guess whether she is observing a communication with S_0 or S_1.

The function α_{RA} works in an analogous way as α_{SA}, but for end-receivers. The only difference is that α_{RA} additionally checks that the provided end-sender S is honest. Indeed, since, by design of our protocol, the end-sender knows the identity of the end-receiver it contacts, it does not make sense to test receiver anonymity in this case. We remark that the α_{SA} and α_{RA} are similar to the (session) sender anonymity and receiver anonymity, as defined by Backes *et al.* [Bac+13].

Finally, the function α_{SU} is more complex, and closer to the notion of *session sender unlinkability* in AnoA[7]. The function α_{SU} takes place in two stages (hence the values $stage1$ and $stage2$ in the state). In the first stage, $\mathcal{A}$ can observe the sending of one or several messages in a session between S_a and R_a with $ocomid$ (a being randomly taken in $\{0, 1\}$). In the second stage, $\mathcal{A}$ can observe another session (with $ocomid'$) with the same end-sender/end-receiver pair ($b = 0$ and $a' = a$), or a different one ($b = 1$ and $a = 1 - a'$). This captures the desired notion of session unlinkability: if $\mathcal{A}$ can not distinguish these two cases, it implies that she is unable to link sessions between the same end-sender and end-receiver. The verifications performed by α_{SU} ensure that the challenge focuses on honest end-senders and end-receiver, and that topology dissemination is terminated.

Theorem 5 formally defines the SA, RA, and SU security properties. The theorem introduces an *adversary class* $\mathcal{A}'$, and states security against $\mathcal{A}'(\mathcal{A})$. In the AnoA framework, an adversary class roughly acts as a wrapper for $\mathcal{A}$, filtering the information it receives, or producing a subset of inputs on its behalf (without an adversary class, $\mathcal{A}$ would the one producing *all* inputs to give to $\mathcal{F}$). Here, we use the mechanism of adversary classes in the second manner. More precisely, $\mathcal{A}$ is only allowed to produce the setup inputs of corrupted nodes, while $\mathcal{A}'$ is responsible for giving the setup inputs

[7]The fact that this property also has the acronym SU is pure coincidence and not meaningful.

of honest nodes. This prevents $\mathcal{A}$ from knowing the honest nodes' *src* and *dst* values. We however allow the adversary $\mathcal{A}$ to produce all the send inputs, even of honest node, which effectively grants her to control all communications in the network. This implies that $\mathcal{A}$ can easily make it so there is only one communication at any time, allowing her to trivial link together (intermediary) corrupted sub-paths.

Theorem 5 (SA, RA, SU). *For any set of nodes Ω among which a random subset $\Omega_c \subseteq \Omega$ of nodes are corrupted, and for any network in which, after topology dissemination, any node can reach any other node with a maximum route length of $l_{max} \in \mathbb{N}$, assuming that end-senders and indirection nodes do not collude, and assuming pseudonym indistinguishability and the IK-CPA property of the Elgamal scheme, $\mathcal{F}$ is $(0, \delta)$-α-ind-cdp for $\alpha \in \{\alpha_{SA}, \alpha_{RA}, \alpha_{SU}\}$,* i.e. *for all adversary $\mathcal{A}$,*

$$\Pr\left[\mathcal{A}'(\mathcal{A})^{Ch(\mathcal{F},\alpha,0)} = 0\right] \leq \Pr\left[\mathcal{A}'(\mathcal{A})^{Ch(\mathcal{F},\alpha,1)} = 0\right] + \delta$$

with $\delta = 1 - \frac{\binom{|\Omega|-l_{max}}{|\Omega_c|}}{\binom{|\Omega|}{|\Omega_c|}} + \mathsf{negl}(\lambda)$ *for SA and RA and* $\delta = 1 - \frac{\binom{|\Omega|-2l_{max}+1}{|\Omega_c|}}{\binom{|\Omega|}{|\Omega_c|}}$ *for SU*

Proof Sketch 5 (Proof Sketch of Theorem 5). The full proof can be found in Appendix B.6. It follows the methodology that Backes et al. *[Bac+13] apply to Tor. The analysis and proof of each property is driven by a* distinguishing event *E. For each property, E is defined so that, if $\neg E$ occurs, the adversary gets the same information whether $b = 0$ or $b = 1$, up to some pseudonyms or encryptions of one. Therefore, we can show that, when $\neg E$ occurs, $\mathcal{A}$ can not distinguish the two cases (up to a negligible advantage, bounded according to the properties of the Elgamal scheme and the indistinguishability of pseudonyms). That is, we show for each property $\texttt{PROP} \in \{\texttt{SA}, \texttt{RA}, \texttt{SU}\}$, that:*

$$\Pr\left[\mathcal{A}'(\mathcal{A})^{Ch(\mathcal{F},\alpha_{\texttt{PROP}},0,)} = 0 \;\middle|\; \neg E_{\texttt{PROP}}\right] = \Pr\left[\mathcal{A}'(\mathcal{A})^{Ch(\mathcal{F},\alpha_{\texttt{PROP}},1)} = 0 \;\middle|\; \neg E_{\texttt{PROP}}\right] + \mathsf{negl}(\lambda) \tag{5.6}$$

Here, the distinguishing event E_{SA} for SA (resp. E_{RA} for RA) corresponds to the presence of at least one corrupted node on the first (resp. second) leg of the challenge communication. For SU the distinguishing event E_{SU} corresponds to the conjunction of: the presence of at least one corrupted node on any leg of the route of the first session between S_a and R_a and *on any leg of the second session between $S_{a'}$ and $R_{a'}$. The equality (5.6) holds perfectly in the case of SA and SU (there is no $\mathsf{negl}(\lambda)$ factor). However, for RA, the equality holds up to a negligible additive factor that depends polynomially on $\mathsf{Adv}^{\mathsf{ps-ind}}(\lambda) + \mathsf{Adv}^{\mathsf{ik\text{-}cpa}}(\lambda)$. This is due to the fact that, during the topology dissemination phase, the corrupted nodes of $\mathcal{A}$ will necessarily get pseudonyms and encryptions of one towards R_0 and R_1.*

*Equality (5.6) holds only when E does not happen. However by the way the distinguishing events are defined, when they occur, there is at least one piece of information leaked to $\mathcal{A}$ that differs depending on the value of b. This difference gives the adversary a possibly non-negligible advantage. Here, we over-approximate this advantage, and assume it to be maximal (*i.e. *$\mathcal{A}$ distinguishes with probability 1 when E occurs).*

Given eq. (5.6) and the advantages of the adversary when E occurs and when it does not occur, it remains only to quantify the probability that this event happens. Subsequently, the theorem is obtained by the law of total probability. This quantification of $\Pr[E_{SA}]$, $\Pr[E_{RA}]$, *and* $\Pr[E_{SU}]$ *yields the δ values specified in Theorem 5. Those are defined over the random choice of $\mathcal{A}$ in corrupting a specific subset of nodes.*

5.6.3.c) Proving MU

The MU property is divided between MU-session and MU-tracing. As already noted, MU-session is simply not fulfilled for oriented communication session, in the presence of corrupted nodes. In particular, a corrupted indirection node trivially breaks it. Therefore, we only prove the MU-tracing property. This is done with the same formalism as for the proofs of the route proposal properties. However, the definition of the adversarial View is slightly modified, to adapt it from a view on route proposals to a view on oriented communication sessions. The term $\mathsf{View}^{\mathcal{N}}_{S\to R,ocomid,init}$ denotes the information leaked by $\mathcal{F}$ to corrupted nodes in $\mathcal{N}$, restricted to information directly relating to the communication between S and R with *ocomid*. The term $\{data\}$ specifies the exchanged payloads. If the term *init* is present, the view also contains the interaction between end-sender and indirection node during the initialisation of the oriented communication; otherwise, only the data exchange is given in the view.

The properties of MU-tracing and of route proposal *return trip untraceability* are somewhat similar. That is, they are properties focusing on *sub-paths*. Thus, the approach to formally define MU-tracing is to *commit* to a communication session between one sender-receiver pair, but only for a *portion* of the route (by restricting the view to a subset of corrupted nodes); and then giving as challenge the view of *different* corrupted nodes during the same communication session ($b = 0$) *or* a different one ($b = 1$). We do, however, make the restriction that the two sessions in question have a portion of route in common, and that the challenge focuses on this portion. Overall, we decide to define MU-tracing on the following dual scenario: either a corrupted end-sender and first leg (possibly including the indirection node) trying to trace messages headed towards to a (honest) end-receiver; or conversely, a corrupted end-receiver and second leg (possibly including the indirection node) trying to trace back a message coming from a (honest) end-sender. This represents the ultimate goal of the MU-tracing property: to conceal where a message goes or where it comes from. Additionally, this encompasses many other scenario, *e.g.* of indirection and relay nodes alone aiming at tracing messages back to an honest end-sender and/or to an honest end-receiver.

Lastly, note that the definition of MU-tracing is *quantitative* (with a non-negligible adversarial advantage), since, similarly to SA, RA and SU, the MU-tracing property does not hold with a negligible advantage under *all* corruption configurations. Namely, MU-tracing on the first leg holds up to a *non-negligible* advantage.

Definition 26 (MU-tracing). *The protocol satisfies (δ_S, δ_R)-MU-tracing if for all PPT adversary $\mathcal{A}$, it holds that:*

$$\forall i \in I_{\mathcal{F}}^{MU\text{-}S}, \Pr[\mathcal{A}(view_0(i)) = 0] \leq \Pr[\mathcal{A}(view_1(i)) = 0] + \delta_S$$

$$\forall i \in I_{\mathcal{F}}^{MU\text{-}R}, \Pr\left[\mathcal{A}\left(view_0(i)\right) = 0\right] \leq \Pr\left[\mathcal{A}\left(view_1(i)\right) = 0\right] + \delta_R$$

where $view_b$ *is defined as follows for any nodes* S_0, S_1, R_0, R_1, *and relay nodes* $\mathcal{Z}_1, \ldots, \mathcal{Z}_K$. *Let* $a \leftarrow_{\$} \{0,1\}$ *and* $a' = a$ *if* $b = 0$ *or* $1-a$ *if* $b = 1$, *and define:*

$$view_b(i) \coloneqq \Big(S_0, R_0, S_1, R_1, \mathsf{Context}_{R_0,R_1,I_0,I_1}^{\mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K}(i), \mathsf{View}_{S_a \to R_a, ocomid_a, init}^{\Omega_c \setminus (\mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K)}(i), \mathsf{View}_{S_{a'} \to R_{a'}, ocomid_{a'}}^{\mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K}(i)\Big)$$

$I_{\mathcal{F}}^{MU\text{-}S}$ *and* $I_{\mathcal{F}}^{MU\text{-}R}$ *are sets of input values for* $\mathcal{F}$ *that both yield executions where* $S_0, S_1, R_0, R_1 \in \Omega$, *where all* Z_{k,i_k} *are corrupted except* $Z_{k\neq 1,1}$ *and* $Z_{k\neq 1,n_k}$; *where* S_0 *initiates a session with* R_0 *with* $ocomid_0$ *using indirection node* I_0 *in order to send messages* $\{data_0\}$, *and, independently,* S_1 *initiates a session with* R_1 *with* $ocomid_1$ *using indirection node* I_1 *to send* $\{data_1\}$; *and where both sessions share a common portion of route, described by intermediary corrupted sub-paths* $\mathcal{Z}_1, \ldots, \mathcal{Z}_K$.

Additionally, $I_{\mathcal{F}}^{MU\text{-}S}$ *(resp.* $I_{\mathcal{F}}^{MU\text{-}R}$*) yield executions where* S_0 *and* S_1 *(resp.* R_0 *and* R_1*) are honest, and* $\mathcal{Z}_1, \ldots, \mathcal{Z}_K$ *represent* all *the* intermediary *corrupted sub-paths from the first (resp. second) leg.*

Theorem 6 (MU-tracing). *Assuming that end-senders and indirection nodes are not colluding, assuming the indistinguishability of pseudonyms, and the IK-CPA property of the Elgamal scheme,* (δ_S, δ_R)*-MU-tracing holds with* δ_S, δ_R *defined as:*

$$\delta_S \leq 1 - \frac{\binom{|\Omega|-2}{|\Omega_c|}}{\binom{|\Omega|}{|\Omega_c|}} + \mathsf{negl}(\lambda)$$
$$\delta_R \leq \mathsf{negl}(\lambda)$$

Proof Sketch 6 (Proof Sketch of Theorem 6). The full proof is provided in Appendix B.6.2. The proof outline consists in making explicit the view, when $\mathcal{Z}_1, \ldots, \mathcal{Z}_K$ *are on the first leg (*$i \in I_{\mathcal{F}}^{MU\text{-}S}$*) and when they are on the second leg (*$i \in I_{\mathcal{F}}^{MU\text{-}R}$*).*

For case of the second leg, it appears that the only information given in the challenge *view* $\mathsf{View}_{S_{a'} \to R_{a'}, ocomid_{a'}}^{\mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K}(i)$ *consist of the pseudonyms* $PS_{Z \to R_{a'}}$ *and encryptions* $c_{\mathtt{one}Z \to R_{a'}}$. *Whereas in the* commit *view* $\mathsf{View}_{S_a \to R_a, ocomid_a, init}^{\Omega_c \setminus (\mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K)}(i)$ *no information is leaked on either* R_0 *nor* R_1 *that would allow to de-anonymise these pseudonyms or encryptions. This holds whether or not the end-sender* S_a *or the indirection node* I_a *is corrupted (recall that by assumption, if the end-sender is corrupted, the indirection node is not, and* vice-versa*). Thus, by the indistinguishability of pseudonyms and the IK-CPA property of the Elgamal scheme,* $\mathcal{A}$ *only has a negligible advantage, that can be bounded above by* $l_{max} \cdot (\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ps-ind}}(\lambda) + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{ik\text{-}cpa}}(\lambda))$.

For the case of the first leg, the situation is different, because the challenge view contains pseudonyms $PS_{Z \to I_{a'}}$, *which* $\mathcal{A}$ *knows to be pseudonyms towards* $I_{a'}$. *Recalling that, ultimately, the goal of* $\mathcal{A}$ *is to know whether* $a' = a$ *or* $a' = 1 - a$, *it is clear that if*

$\mathcal{A}$ succeeds in distinguishing whether $I_{a'}$ (as seen in the challenge view) is the same as I_a, then she wins. From this, it can be shown that, in the distinguishing event *that one of I_0 or I_1 or both are corrupted, the presence (or absence) of leaks such as* ocom-I *in the commit view allows $\mathcal{A}$ to distinguish the view with probability one. The probability that I_0 or I_1 or both are corrupted being $1 - \binom{|\Omega|-2}{|\Omega_c|} / \binom{|\Omega|}{|\Omega_c|}$, the value of δ_S can be shown following the same methodology as the (full) proof of Theorem 5 (which deals with SA, RA, and SU).*

5.7. Summary

In addition to the summary of results and insight presented in Section 5.2, the following remarks wrap up this chapter on the formal treatment of the protocol.

Results and Lessons Learned

We showed formal proofs for each properties of the protocol, in a very constraining model. The first step of proofs, with the UC framework, led to the expected ideal functionalities, in particular leaking *corrupted sub-paths* and dst^{src} values, but leaking no information at all to network observers. However, during the analyses of these functionalities with the AnoA framework, we were only able to obtain under-approximation of the actual anonymity provided by the protocol. Either because an in-depth analysis would have been too complex and lengthy (even for a thesis), either because the formal foundations to carry out these analyses were plainly missing in the state-of-the-art. As a result, several interesting features of the protocol can not be integrated in the proofs. In particular, the fact that end-senders share portions of routes, or that oriented communication initialisations use `rtproprelay` messages indistinguishable from those of a route proposal return trip, and most of the elements of design that aim at preventing traffic analysis, have not played a part in the proofs.

More generally, it can be said that, as of today, the cryptographic frameworks focusing on the formal study of communication networks do not seem to deal well with the countermeasures pertaining to traffic analysis. Said otherwise, it is unknown how to quantify the adversarial advantage obtained from traffic analysis in a *provable secure* way (with an upper bound that encompasses the worst possible cases). This implies that, to still carry out formal proofs, one has to make the conservative assumption that gives her full advantage (or alternatively, to pose strong and possibly unrealistic assumptions).

Still, this work is a first step, which makes the basic properties of the protocol appear. Above all, it shows that *under heavy constraints*, with a global observer and corrupted colluding nodes, even if the traffic load is extremely low (only one communication at a time), anonymity can be achieved. In comparison, the cMix protocol [Cha+16] provides comparable privacy guarantees, but is a synchronous mixnet, which by definition requires many users to communicate at the same time to *fire* the mixes[8]. And the Tor

[8] Or more exactly, the network must wait for enough end-senders to be ready to send

protocol [DMS04] simply does not provide any anonymity against a global external adversary.

Going Further

There are *gaps* in the chosen approach for proving MU and the route proposal mechanism: the proofs hold only for a subset of all possible system executions. More generally, it would be preferable to have a *standard* and widely accepted security definition for MU and the route proposal mechanism. In particular, the MU property is present in the literature, but, to the best of our knowledge, has never been formally proved.

Secondly, the proofs of SA, RA and SU are *loose*, meaning that the anonymity may actually be proved to be better. It would be interesting (although extremely cumbersome) to delve into more details, in order to obtain tighter and more accurate bounds. One possible lead to do so is to follow the ideas of Backes *et al.*'s more recent works on Tor [BMS16], which takes into account different strategies for choosing the relay nodes of circuits. Note that this would require assumptions on the topology graph and the route proposal policy. Indeed, the topology graph not being complete, the identities of previous and next hop nodes leak information on the potential end-sender and end-receivers to corrupted nodes on the route. In addition, this analysis would require to define a concrete route proposal policy. Indeed, the route proposal policy determines the probability that a given route is effectively created in the network. Thus, for a corrupted relay node on a route, the route proposal policy determines the potential end-senders (and end-receivers), and their probability to be the actual end-sender (resp. end-receiver). Formally taking into account these elements in a detailed analysis is a challenge in itself. However, in the next chapter, we make a step towards studying the impact of these elements on anonymity. We empirically measure the distribution of routes length, and model collusions of corrupted nodes on the routes trying to find the end-sender or end-receiver of a communication, by enumerating all the nodes reachable in less than a certain number of hops in the topology graph. As a result, we obtain an empirical quantification of SA and RA, showing much more *optimistic* results than the level of anonymity proved in this chapter.

Finally, of course, the proofs must be ported to the adaptive, fully malicious adversary model. For that, heavy modifications are needed on the protocol.

6. Implementation: Practical Performances and Privacy

After the formal and theoretical security analysis, this chapter studies the protocol under a more practical point of view. It presents a *proof-of-concept* implementation of the protocol, and puts in light different properties related to its general efficiency and privacy. It also studies the route proposal, message re-ordering, dummy messages and controlled traffic rates mechanisms. Finally, in complement to the theoretical privacy quantification from the previous chapter, an empirical measure of privacy is presented.

6.1. Implementation Choices

The motivation behind the proof-of-concept implementation is before all to evaluate the time taken by topology dissemination, the latency in message delivery, and the privacy provided by the protocol. The implementation is not meant to account for the physical network phenomenons, nor to study how the protocol performs over a real-world transport protocol such as TCP/IP.

The protocol was implemented using a *discrete-event simulator* [WLW09], as it is common for a first protocol implementation. Here, we chose to use *SimPy* [Mul+], a generic-purpose simulator written in Python. The protocol implementation is thus also written in Python. SimPy is not dedicated to *network* simulations, and does not provide any network model (*e.g.* wireless mesh network, Internet overlay). It merely consists in a generic API for discrete-event simulation, which is sufficient for our needs. Indeed, in the proposed protocol, nodes' behavior is driven by the batching rounds, *i.e* nodes *activate* every $t_{\mathcal{P}}$ seconds to process received messages and send out others to their neighbors. Thus, the main events needed for the simulation are simple *delay* events of $t_{\mathcal{P}}$ seconds, which SimPy handles perfectly.

Transposing the abstract protocol description from Chapter 4 into an implementation requires making certain choices, such as the exact route proposal policy that nodes are

going to follow. Below are listed some of the most relevant points of implementation. For more details, see the publicly available code of the implementation [Gue17].

Network We used the NetworkX library [Net] to generate and manipulate the topology graph. Before running the network, a random regular graph is generated, where each node has exactly $\log |\Omega|$ neighbors. The resulting graph is connected and undirected.

Cryptography We did not implement the cryptographic operations. First, since these operations are known and proved to be secure, implementing them would not allow to further attest their security. Secondly, even though cryptographic operations (especially modular exponentiations) are known to be time and resource-consuming, the times required to encrypt or decrypt with AES or Elgamal, or to compute homomorphic operation, are expected to be negligible compared to the batching time interval $t_{\mathcal{P}}$. However, in order to check the correctness of the homomorphic operations used in the network (in particular concerning the accumulation of temporary keys during a return trip), we implemented basic group operations.

Batching, Dummy Messages, and Controlled Traffic Rates In the Python implementation, message re-ordering, dummy messages, and controlled traffic rates are implemented exactly as described in Section 4.4.2, with two exceptions. First, by the way the SimPy implementation works, the network is actually *synchronous*, *i.e.* all nodes are always in the same batching round. Secondly, we introduce a *sampling bias* when choosing messages from the pools, so as to select real messages more often than dummy ones. That is, messages to send in each round are not selected according to a uniform distribution pools, but biased towards real messages. Indeed, from preliminary measures without this bias, it appeared that real messages stayed a very large number of rounds in the pools before being selected, thus greatly degrading the performances of the protocol. In practice, we programmed this bias to depend on the node's allowed sending budget in each given round (that is, the more real messages a node is allowed to send as per the traffic rates constraints, the greater the bias is). In Section 6.3, we show that this design choice does not severely impacts privacy, and the TAR property in particular.

Concrete Route Proposal Policy Finally, a crucial part of the implementation consists in proposing a concrete route proposal policy. Indeed, Section 4.5.4 merely discussed the route proposal policy, leaving this element of design as an implementation choice. In this proof-of-concept implementation, the guidelines exposed in Section 4.5.4 are followed, and the route proposal policy exhibit the following features

- Route length limitation, using the described encrypted decision process. This requires the return trip from proposee to receiver to carry two link messages (one with dst^{src}, the other with $g^{l_{max}-l}$).
- Possibility for a node to have several routes towards the same receiver (*i.e.* towards the same pseudonym), up to a maximum max_{rt}.

- Connectivity: ensure that, at the end of topology dissemination, each node has a route towards each other.
  - Always accept the first route proposal towards a newly discovered pseudonym.
  - Make each node propose each other node at least once (here, *exactly* once).
- Unpredictability of created routes: introduce probabilistic behavior.
  - A node accepts a route towards a pseudonym it already knows on a random basis, according to $p_{reaccept}$
  - For such an accepted route, a node can randomly choose to replace a route it already knows with probability $p_{replace}$, or to add a new route with probability $1 - p_{replace}$.
  - Avoid shortest paths: do not immediately re-propose the first route learned towards a pseudonym (wait to see if a second route is learned, and propose that one preferably).

The route proposal policy parameters l_{max}, max_{rt}, $p_{reaccept}$, and $p_{replace}$ are tuned as follows. First, the maximal route length l_{max} is automatically set according to the longest among all shortest paths between pairs of nodes in the topology graph. Then, max_{rt} is arbitrarily set to 3, for it leaves enough choice to node, and yields routing tables of reasonable size. For the $p_{reaccept}$ and $p_{replace}$ parameters, we ran several simulation of the network with various values, in order to measure the *unpredictability* of obtained routes. This *unpredictability* is measured as the *Jaccard distance* between the set of routes created in one network run, and those created in a second network run (on the same topology graph). This distance metric is defined over two sets of items A and B, not necessarily of the same size, as $(|A \cup B| - |A \cap B|)/|A \cup B|$. A Jaccard distance of 1 means that the two topology dissemination runs did not produce a single route in common. This is abstractly what is aimed at for *unpredictability*, since this means that the set of created routes is very different between each topology dissemination run on the same graph. In addition to the route unpredictability, we also measured the effective number of routes per nodes, in order to attest that each node obtains its three routes towards each other node. The results of these network simulations are shown in Fig. 6.1 and 6.2. Although the other parameters of the protocol, pertaining to the message re-ordering mechanism in particular, may have had an influence on the results, this was not measured. Here, simulations were run using the *standard* set of parameters $f_{\mathcal{P}} = 0.5$, $n_{\mathcal{P}min} = 5$, $f_{dum} = 0.8$, $\Delta r = 8$, described in details in the next section.

The results interestingly show that, even with $p_{reaccept} = 0$ and $p_{replace} = 0$, the route unpredictability is not zero. That is, the network naturally ensures a *base value* for route unpredictability. This is largely imputable to the pool-based mixing of messages, that introduces randomness in the order messages are delivered, and thus in the order that route proposal are carried out. However, it is desirable to reach a higher Jaccard distance than this base value. Here, we choose to set $p_{reaccept} = 0.5$ and $p_{replace} = 0.25$, for these are the lowest values achieving a high route unpredictability of 0.90 or above, and which

still provide a relatively high effective number of routes between any two pair of nodes (2.5 in average).

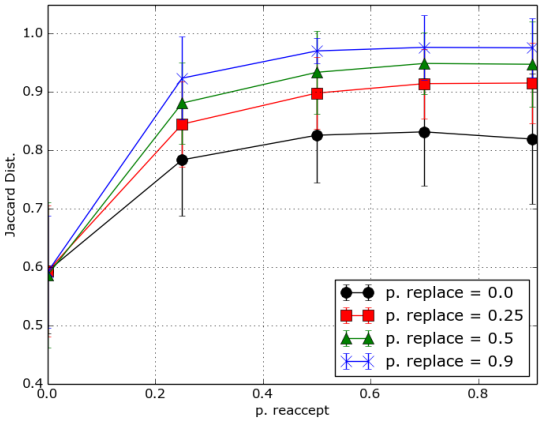

Figure 6.1. – Routes *unpredictability*

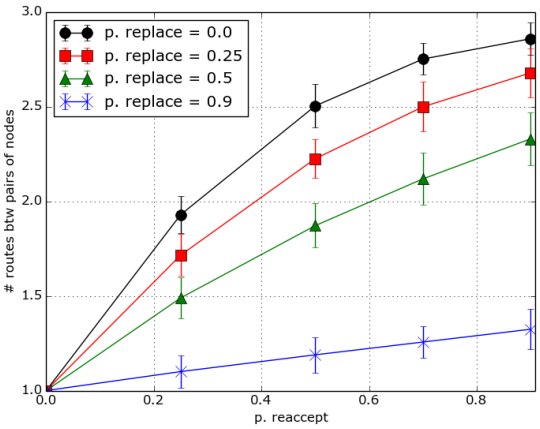

Figure 6.2. – Number of routes

6.2. Efficiency

The first motivation behind the implementation is to test the performances of the network, and to study the effects of the different protocol parameters. Of particular interest here are the parameters of the batching, dummy messages, and controlled traffic rates mechanisms (namely, $n_{\mathcal{P}min}$, $f_{\mathcal{P}}$, f_{dum} and Δr), and their influence on the time needed to disseminate the topology, and to deliver payload messages in oriented communication sessions. To that end, network simulations with $|\Omega| = 100$ nodes were run. Topology is fully disseminated in a first phase, and oriented communications take place in a second phase. In the later, each node chooses 5 receivers at random, to which it sends 40 payload messages each. For each of these five receivers, the node also picks a random (possibly different) indirection node, with which it runs the oriented communication initialisation.

Note that the batching time interval parameter $t_{\mathcal{P}}$ is not tested. Indeed, the network being synchronous, the value of the time delay between rounds has actually no impact on the performances of the simulated network[1]. Therefore, we measure network performances in number of rounds, allowing a generic expression of results. To interpret them, we will however consider that $t_{\mathcal{P}}$ is equal to one minute.

Table 6.1 shows the tested parameter combinations, and the results. The first table row gives the result for the parameter set $n_{\mathcal{P}min} = 5$, $f_{\mathcal{P}} = 0.5$, $f_{dum} = 0.8$ and $\Delta r = 8$, hereby denoted the *standard* parameter set. Note that the values for the batching parameters $n_{\mathcal{P}min}$ and $f_{\mathcal{P}}$ are in accordance with the values proposed in the Mixminion implementation by Nick Mathewson [Mat11]. Subsequent table rows show results for alternative parameter combinations. A "−" symbol indicates a parameter left to its

[1] At least as long as $t_{\mathcal{P}}$ is set sufficiently large in comparison to the expected time needed for nodes to process and send messages in a round.

standard value. While alternative values for f_{dum} and Δr were independently tested, $f_{\mathcal{P}}$ and $n_{\mathcal{P}min}$ were tested together, for they are strongly tied together in the pool-based mixing mechanism. For each parameter combination, Table 6.1 shows the two main performance metrics: the number of rounds necessary for topology dissemination to complete, and the amortized number of rounds for one oriented communication message to be delivered from end-sender to end-receiver, *i.e.* the latency of the entire session, divided by the total number of messages in the session. The latter metric is presented with and without taking into account the oriented communication initialisation into account (numbers in brackets are *with* initialisation). The presented figures are averages and standard deviations over several network runs. The table gives additional metrics that help towards interpreting the results: the ratio of the overall number of real messages that were sent in the network over the number of dummy ones; the average number of rounds between the insertion of a real message in a pool and its actual sending (*pool delay*); and the percentage of rounds in which a node has to resort to using end-to-end dummy messages in order to respect the controlled traffic rate equation.

Parameters	***Topo. Diss.*** (rounds)	***1 Ocom Msg*** (rounds)	***# reals / # dum*** (msg. ratio)	***Pool Delay*** (rounds)	***E2e dum.*** (round ratio)
$f_{\mathcal{P}} = 0.5$ $n_{\mathcal{P}min} = 5$ $f_{dum} = 0.8$ $\Delta r = 8$	1519.80±124.50	14.59±0.22 (18.61±0.40)	42.44%±1.65%	15.15±5.94	0.26%±0.03%
$f_{\mathcal{P}} = 0.8$ $n_{\mathcal{P}min} = 2$ – –	1610.60±144.21	14.86±0.45 (19.03±0.55)	36.67%±1.09%	16.91±6.17	0.18%±0.03%
– – $f_{dum} = 0.3$ –	2891.80±119.96	25.13±0.68 (32.30±0.86)	26.58%±0.80%	43.91±9.62	0.07%± < 0.01%
– – – $\Delta r = 2$	1803.80±164.28	18.54±0.55 (23.45±0.61)	45.46%±2.65%	15.31±6.47	2.14%±0.29%

Table 6.1. – Network Performances for $|\Omega| = 100$ nodes

Results for the standard parameter set show that, with $t_{\mathcal{P}} = 1min$, topology dissemination takes slightly more than 24 hours. This represents a full day, before being able to start sending messages. However recall that topology dissemination is meant to be run only once. Then, the amortized delivery latency of one payload message, from end-sender to end-receiver, is approximately 15 minutes (and 18 minutes when taking the oriented communication initialisation into account). This is an acceptable delay for the envisioned application, and for a high latency protocol in general. A very large part of the introduced delays can be explained by the batching strategy and the controlled traffic rates mechanisms, as the remaining columns of Table 6.1 show. Indeed, out of all messages sent in the network, only 42.44% are real ones, *i.e.* nodes spend more than half of their time sending dummy messages. Then, per the batching strategy, a real message is typically delayed for 15 rounds at each node. This means that, on a route of length l,

a message takes l times 15 rounds in average to be delivered[2]. This is however the cost for resisting traffic analysis. On the other hand, for the standard parameters, a node needs only rarely (in only 0.26% of rounds) to resort to end-to-end dummy messages in order to respect the traffic rates constraints, which was the expected behavior.

The other parameter combinations shows that performances degrade when f_{dum} and $n_{\mathcal{P}min}$ decrease. In particular, a lower f_{dum} yields a particularly inefficient network. It seems that a consequence of decreasing f_{dum} (and thus inserting less dummy messages in neighbor pools) is that there are not enough messages in pools to *fire* the mix at every round. Indeed, a node can only sample and send messages from the pools if $\min(n_{\mathcal{P}_{Y_i}} - n_{\mathcal{P}min}, n_{\mathcal{P}_{Y_i}} \cdot f_{\mathcal{P}}) > 0$ for each neighbor Y_i (see Section 4.4.2). Then, a low value for Δr decreases performances, mainly because nodes need to adjust more frequently their dummy budgets to respect the traffic rates constraints, and this is realised by sending more link or end-to-end dummy messages instead of real ones. Lastly, parameters $f_{\mathcal{P}} = 0.8$ and $n_{\mathcal{P}min} = 2$, although they are more *permissive* (since they allow to send almost all the messages contained in pools in any given round), surprisingly do not imply better performances. In fact, they are comparable to the performances of the standard parameter set. More simulations would be needed to understand the impact of these parameters on the overall efficiency.

These results do not show how the network behaves in function of its size. Intuitively, one would expect that the time required for topology dissemination (in particular) depend quadratically on the number of nodes $|\Omega|$, since $|\Omega|^2$ routes must be created. However, from complementary results (not depicted in Table 6.1), it seems that the network scales well: the topology dissemination latency only augment linearly or sub-linearly with the number of nodes in the network. For instance, for 70 and 50 nodes respectively, the number of rounds for topology dissemination is roughly 1275 and 1068. As for the amortized latency of one oriented communication message, it actually *decreases* as the number of nodes augments: for 70 or 50 nodes, this latency is respectively of 15.35 and 16.03 rounds. This scaling effect can be explained by the increase of the number of neighbors per node, and the overall augmentation of connectivity and traffic, which allow real messages to be relayed faster. Indeed, the *pool delay* increases to 16.76 and 18.16 rounds for 70 and 50 nodes respectively. A second factor that comes into play is the route length, that we have measured to be approximately $\log |\Omega|$ for the chosen type of topology graph. This means that the route length, and thus the amount of work required to create a single route, augments only logarithmically with the number of nodes in the network.

6.3. Traffic Analysis Resistance

Section 6.1 already studies the *routes unpredictability*. A high route unpredictability intuitively works towards TAR, since it prevents the adversary from knowing with

[2]Note that this is consistent with the latency of an oriented communication message, since the later is presented *amortized* over 40 messages sent in parallel.

certainty the created routes. However, as mentioned in the same section, we introduced a *sampling bias* in the random selection of messages from neighbor pools, in order to favor the selection of real messages over dummy ones, and increase the network efficiency. This *bias* tampers with the traditional functioning of pool-based mixing of messages. Indeed, the idea of this mechanism, in addition to re-ordering messages, is to delay each of them independently: each message may stay an undetermined (and theoretically unbounded [SDS02]) number of rounds in a pool before being selected for sending. This introduces uncertainty regarding the mapping of incoming and outgoing messages of a node, since (contrarily to simpler mixing strategies) a message entering a node at round r does not necessarily comes out from this node at round $r + 1$, but at round $r + r'$ for some r'. The value r' is what we call the *pool delay*. Intuitively, a higher *uncertainty* regarding the value of r' implies a better resistance to traffic analysis. Yet, the bias we introduce tampers with the functioning of pools, and may lower that uncertainty. This section thus studies this aspect.

From Table 6.1 in the previous section, it can already be noted that the pool delay of real messages exhibits a large standard deviation: 5.94, for a mean of 15.15 (for standard parameters). Informally, this shows that the pool delay of each message is indeed hard to predict with certainty. This tendency is confirmed by Fig. 6.3, which shows a detailed view of the pool delay per message. This figure takes the form of a probability distribution, obtained from an empirically measured histogram over several network runs. The maximum delay being greater than a thousand rounds, the figure cuts the histogram to $r = 30$ on the x axis, and the probabilities for delays over 30 are stacked together. This shows that the probability that a message is delayed for 30 rounds or more is far from negligible, since it is approximately 0.14. Actually, the results fit well into a log-normal distribution with parameters $\sigma = 1.60$ and $\mu = 0.93$. This distribution is *heavy-tailed*, implying in our case that the probability that a message stays a large number of rounds in the pools is low, but not *exponentially small*. It also means that the distribution has high entropy (namely, 4.91), intuitively showing that predicting the pool delay for one message is *hard*, which is what is desired for privacy. And at the same time, note that a message has more than 50% chance of spending only five rounds or less in the pools, which yields good efficiency.

It thus seems that the pool-based mixing mechanism, even with the sampling bias, introduces enough uncertainty regarding the per-message pool delay, while actually naturally favoring low delays. This result, along with the results on route unpredictability, are arguments towards showing that the protocol is resistant against traffic analysis.

6.4. Privacy

In addition to measuring network performances, the goal of this chapter is to study the privacy provided by the network on empirical grounds. In particular, we are interested in confronting the obtained quantification of SA and RA from the formal proofs, to a less conservative, empirically measured value. It is assumed here, in contrast with the model of formal proofs, that even corrupted nodes can not perform traffic analysis.

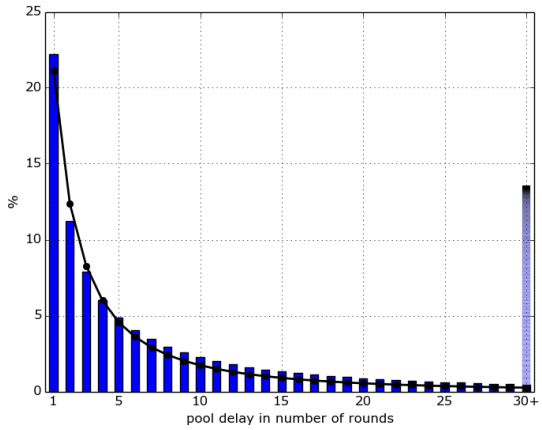

Figure 6.3. – Probability Distribution of Pool Delays

6.4.1. Proposed Methodology

In this thesis, we choose to quantify privacy using an approach inspired from Tarzan (itself inspired from Crowds). It is based on the argument that corrupted nodes on a route can estimate their distance, in hops, to the end-sender or end-receiver. From this, and assuming that she knows the topology graph, the adversary can deduce a set AS of potential end-senders or end-receivers. Tarzan measures anonymity by the *confidence* that the adversary has in de-anonymising the end-sender or end-receiver, computed as $1/|AS|$. This approach implicitly assumes that the adversary assigns the same probability to each potential end-sender/end-receiver in AS.

We adapt this methodology to our setting of oriented communications, taking it one step further: we assume that the probability distribution of routes length is known to the adversary, and that she can deduce a *non-uniform* probability distribution on the anonymity set. From this distribution, we take out the most probable end-sender (or end-receiver), and consider it to be the adversary's guess. Repeating this process for many oriented communication routes, we measure the frequency at which the adversary guesses correctly, and consider it to be the anonymity quantification. Here, we only report the results for *end-receiver* anonymity, because the method we use (described below in more detail) actually returns the same results for end-senders and end-receivers.

In detail, here is how the results were acquired. We ran network simulation with the *standard* parameters, where each node sends 40 payload messages to 5 random end-receivers. For a given network simulation, we apply the following processing to each oriented communication route used by the nodes during the simulation. First, we look at the corrupted nodes on the route (node corruption is determined at network startup, according to a corruption ratio parameter), and select the largest collusion of consecutive corrupted nodes on the route. Let's assume that in a 10-node route from X_1 to X_{10}, nodes X_2, X_3, X_6, X_7, and X_8 are corrupted. In this case, it is the collusion $\{X_6, X_7, X_8\}$ that will be retained, and the considered collusion *controls* four hops in total (including the edges to X_5 and X_9). From this, the adversary can deduce that the maximal number

of remaining hops towards end-receiver is $l_{rem} = l_{max} - 4$. Assuming that the probability distribution on routes lengths is known to the adversary[3], she can deduce the probability distribution of *remaining* hops towards the end-receiver. Denote $p(l)$, for $l \in [1, l_{rem}]$, the probability that l hops remain, *i.e.* that the end-receiver is exactly l hops after X_8. The next step, for the adversary (assuming she knows the topology graph), is to construct the sets $AS_R(l)$ of potential end-receivers for each $l \in [1, l_{rem}]$. From there, the adversary assigns to each reachable end-receiver R the probability:

$$p(R) = \sum_{l \in [1, l_{rem}] s.t. R \in AS_R(l)} p(l) / |AS_R(l)|$$

At the end of this process, we obtain the *non-uniform* probability distribution on end-receivers from the adversary's point of view. Finally, we take out the most probable end-receiver (if several receivers have the same maximal probability, one is taken at random), representing the adversary's guess. It is compared to the actual end-receiver, and the adversary's success or failure in guessing the end-receiver is logged.

This approach assumes that the distribution of routes lengths and the topology graph are known to the adversary. On the other hand, it implicitly assumes that traffic analysis is impossible. This is in opposition with Tarzan's approach. Because Tarzan does not aim at ensuring TAR, in the above example route, it would have considered that all hops between the outermost corrupted nodes X_2 and X_8 are controlled by the adversary. The analysis would consequently have been carried out with $l_{rem} = l_{max} - 8$, necessarily yielding an attack with better accuracy. In this chapter, we choose to assume perfect traffic analysis resistance, in contrast with the formal proofs in the previous chapter.

The novel way of measuring anonymity that we propose, adapted from Tarzan, is related in some respects to entropy-based metrics [Daz+03; THV04], that measure anonymity according to the entropy of the adversary's probability distribution on the potential end-senders or end-receivers. In particular, both the entropy approach and ours necessitate assumptions on the *attack strategy* of the adversary, which is in contrast with the *black-box* model favored in formal cryptographic proofs. Still, we argue that the metric we propose better renders the practical anonymity of nodes than those proposed in the past. Indeed, as also noted by Syverson [Syv09], entropy-based measures often fail to reflect the notion of anonymity that actually matters for the network users. This is especially true for anonymity quantification with Shanon's entropy. Indeed, as pointed out by Tóth *et al.* [THV04], a probability distributions may exhibit a high Shanon entropy, but at the same time associate to the actual end-receiver a probability much larger than to all the others. Our approach does not suffer this shortcoming, since it always considers the most probable end-receiver as being the adversary's guess, even if that end-receiver does not especially *stand out* from the others (*i.e.* even if its associated probability is close to that of the other end-receivers).

[3] In practice, we measure this distribution empirically, by looking at routes created in several network runs on the same topology graph.

6.4.2. Results

Figure 6.4 shows the adversary's probability in correctly guessing the identity of the end-receiver (*i.e.* her probability of breaking RA), according to the corruption ratio $c \in [0, 1]$. Each data point shows an average of this probability over several network runs with a particular c value. The theoretical bound from the previous chapter is also included, for comparison. It follows the formula $\delta = 1 - \binom{|\Omega|-l_{max}}{|\Omega_c|} / \binom{|\Omega|}{|\Omega_c|}$, *i.e.* the probability that SA or RA *does not* hold according to the formal proof from Section 5.6.3.

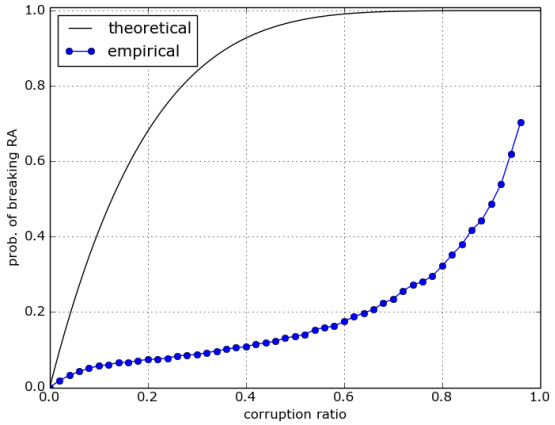

Figure 6.4. – Probability of Breaking SA or RA: Theoretical vs. Empirical

Although our quantification of anonymity is more constraining than previous measures, results show a low probability of success for the adversary, under 0.2 even when there are 60% of corrupted nodes in the network. Also, even when 90% of the network is corrupted, the adversary has only one chance out of two of correctly guessing the end-receiver. Results are not shown for $c > 0.95$, because it is not possible to obtain data points for corruption ratios close to one. Indeed, we measure RA only when the end-receiver is honest, since it does not make sense to *de-anonymise* a corrupted node. Yet, for $c > 0.95$, it often happen that all the oriented communication routes in the network have a corrupted receiver.

Note that these are results for *small* networks of 100 nodes. The fact that the anonymity level is high even for such small networks is actually encouraging. Indeed, this ensures a satisfactory anonymity level even for small networks, and for networks in early stages of development. Also, anonymity only gets better as the number of nodes in the network augments. For a constant corruption ratio, the theoretical anonymity indeed increases with the network size (this is easily verified using the formula for theoretical anonymity), before reaching a stationary level for a few thousand nodes. Also, additional measures show that for 50 and 70 nodes, empirical anonymity is slightly lower than with 100 nodes. For instance, the probability of breaking RA for $c = 0.3$ is 0.099 for 50 nodes, 0.093 for 70 nodes, and 0.088 for 100 node. For $c = 0.7$, these probabilities are respectively 0.274, 0.245, and 0.235. Intuitively, this effect can be explained by the augmentation of neighbors per node, meaning more connectivity, and thus bigger anonymity sets.

Figure 6.4 also allows to compare the theoretical and empirical anonymity levels, and shows great discrepancies between them. Indeed, in formal proofs, SA and RA are considered broken as soon as there is one corrupted node on the route. This is in particular due to the fact that, in the formal analysis, traffic analysis is assumed possible for the adversary, while it is not in the empirically measured privacy. Also, the disparity is explained by the fact that the theoretical anonymity is a lower bound, obtained from a relatively *simple* analysis that does not delve into details. The empirical results show that, in practice, the anonymity level provided by the protocol may be much higher than this theoretical lower bound.

Finally, note that our theoretical anonymity (and, necessarily, our empirical one) is still better than the results from a recent analysis of Tor [BMS16], even though our adversary model is stronger than Tor's. The authors of this study show that, for 20 corrupted Tor nodes, and for the current Tor relay selection algorithm, the adversary's probability in compromising RA is roughly 0.25. Yet, in 2015 at the time this research was conducted, there was more than 6000 such Tor nodes [4]. This means that these results hold for a corruption ratio of $c = 0.33\%$. In comparison, for that corruption ratio, our theoretical and empirical results regarding the probability of breaking RA are lower than 0.05 and 0.01 respectively.

6.5. Concluding Remarks

In this chapter, we presented a proof-of-concept implementation of the protocol. Experimentations show that the protocol introduces high latency, but provides strong privacy. These results indicate that the protocol is suited for the communication of any data in a *non-interactive* manner (*i.e.* for communications that do not necessitate frequent back and forth exchanges), including the exchange of emails, similarly to mixnets. The sending of data can be done in parallel, by sending many messages in a same oriented communication route, and by opening several such routes, thus amortizing the overall delay of data transmission. Note that the use of multiple oriented communication channels in parallel does not degrade privacy, since the SU property ensures a clear separation between the multiple sessions.

This way of using the network fits the informant-journalist scenario, where it is assumed that the informant needs to send a set of documents to the journalist, without requiring an immediate answer from the latter. Also, the fact that the protocol provides strong guarantees even for *small* networks is in accordance with the envisioned use-case. Indeed, even though the inefficiency of the protocol may not provide incentive for many users to join in, we have shown that privacy can still be strong for a small set of users that are willing to pay the cost for privacy.

[4] https://metrics.torproject.org/networksize.html

Conclusion and Perspectives

Over the last years, privacy in Internet communications has been rising as a major concern. Recent events show in particular the necessity of ensuring the privacy of individuals that take personal risks to inform the general public of illegal or immoral actions conducted by governement bodies, such as citizens in an authoritarian regime or so-called *whistle-blowers*. The aim of this thesis was to study the possibility of providing anonymity guarantees suitable for such individuals. For that, we proposed a homogeneous protocol, and studied its security and performances. This protocol can be seen as an attempt at providing the highest level of privacy in a homogeneous network, while still being manageable for hundreds to thousands of nodes, and taking into account a large part of know (passive) attacks of all kinds. We show that it is possible to achieve a level of privacy where the very fact that a node communicates is concealed, for a cost in terms of efficiency comparable to existing *high latency* protocols.

Summary of Contributions

(Formal) Definition of Anonymity Properties

First of all, we defined what *strong* privacy means in our envisioned application, specifying in particular that sender and receiver anonymity should imply the impossibility to detect the very fact that a node communicates (a property also denoted as *unobservable* communications). In Chapter 5, we then studied these properties, and provided formal definitions for each of them.

Survey of Related Works

In our survey of existing works, we underlined the necessity of a homogeneous network architecture to provide the anonymity level we aim for. Also, through a review of known attacks against existing protocols, we were able to identify traffic analysis as one of the main (yet least understood) threats to anonymity.

Design of an Anonymous Protocol

The core contribution of this thesis is a new homogeneous protocol that realises *strong anonymity*, making it suitable for an informant-journalist type of scenario. The originality of our work mainly relies in the use of relationship pseudonyms, that, in particular, has led us to design a *stateful* protocol. That is, a protocol which requires a topology dissemination phase, and which constructs long-lived circuits. This is in opposition to the more traditional approach, in which a sender constructs a circuit only when needed

and uses it only for one particular communication. In the design of our protocol, we also showed one possible way to adapt message re-ordering techniques from mixnets (pool-based ones in particular), into a fully decentralised network, where there is no more distinction between *users* and *mix servers*. Then, we conducted a thorough analysis based on the point of view of a given honest node, which allowed us to formulate equations governing traffic rates and the use of dummy messages, so as to make each node *appear as a simple relay*. This mechanism builds upon the Tarzan protocol, but proposes stricter rules, and is made to resist even against network observers and corrupted neighbors. Another contribution resides in the way we implemented the topology dissemination and the construction of routes. Using homomorphic encryption, we showed how virtually any information on the routes can be computed while minimising the amount of leaked information (in particular, about the identity of the nodes on these routes). Finally, our protocol, to the best of our knowledge, is the first to study the use of relationship pseudonyms in anonymous networking. The cryptographic properties of these pseudonyms allow to realise a functional equivalent to Tor's hidden services, which, in our motivating use-case, allows the informant to stay anonymous even from the journalist. Another benefit of these pseudonyms is the *separation of knowledge* between nodes. Actually, using such pseudonyms is a way to admit that there will always be successful de-anonymisations attacks on the protocol, and to pro-actively limit their impact.

Validation of the Protocol

We validated our protocol in two ways. First, we formally proved its privacy properties using state-of-the-art cryptographic proof frameworks, thereby providing a complete formal analysis of the protocol. Secondly, we implemented a proof-of-concept version of the protocol, and measured its performances and practical privacy. In terms of efficiency, these results show that the protocol introduces high latency in communications, to an extent comparable to existing mixnets. The formal proofs provide results that represent the *worst-case* level of privacy that the protocol achieves. Practical measures of privacy are more optimistic, and show that, even for a small network, anonymity is strong in practice.

Lessons Learned

During our work, we were also able to establish or confirm the following results, that were not initially linked to the main subject of the thesis. First, we noted that pool-based message re-ordering techniques do not only participate in preventing traffic analysis, but also introduce uncertainty regarding all events in the network. In our case, it allows to introduce uncertainty regarding the obtained routes after topology dissemination. Secondly, during the production of formal proofs, we noted that, with today's existing cryptographic frameworks, analysing a protocol as complex as the one we propose is extremely challenging. More accurately, the existing tools allow an analysis of the cryptographic parts of protocols, but do not allow to model traffic analysis or concurrency among network events. Consequently, all the mechanisms that aim at ensuring resistance

against traffic analysis can not be taken into account in proofs. Also, the fact that the protocol is *stateful* is another obstacle towards analysing it thoroughly, since, to analyse one communication, it is necessary to take into account all that happened before that communication, and in particular during topology dissemination. Therefore, although they provide a good basis for a preliminary analysis of our protocol, these cryptographic proofs frameworks are mainly suited to the analysis of simpler protocols, such as (parts of) low latency networks.

Assessment

On the one hand, the popular networks deployed today over the Internet (*e.g.* Tor [DMS04] or I2P [I2P]) aim at providing *anonymity for the masses* [Lin16b]. As such, they feature high efficiency, in order to be usable on a daily basis. These networks however provide protections only against the receivers of communications (*e.g.* web servers) and against network operators, but not against the type of adversary considered in this thesis.

On the other hand, our protocol is more suited for communicants in need of strong privacy guarantees, which are willing to take their time to be cautious in order to avoid being detected. Note that it took more than five months to Edward Snowden between its first contact with the outside and the publication of its revelations [Lee14]. Also, our protocol is tailored for a small (but worldwide) community of privacy activists willing to take part in the network to provide protection to informants. Indeed, our results show that, even when there are only a few hundred nodes in the network (*e.g.* only a few journalists, human rights organisations, and privacy activists), the protocol provides suitable anonymity. Moreover, this *community* aspect appears directly in the design of the protocol, since privacy mainly stems from the willingness of nodes to help each other in staying anonymous, through the mutual provision of cover traffic.

Future Works and Possible Improvements

Considering Malicious Adversaries

The first and foremost improvement to the protocol should be to include protections against malicious behaviors (*i.e. active* adversaries). In particular, there are *active variants* for most of the attacks presented in Chapter 3.

A malicious adversary is assumed to be able to delay, replay, inject, or drop messages flowing in the network. There are various ways in which the adversary can make use of these capacities to degrade privacy [BPS01; SDS02; DMS04; SW06; HB13; Gha16]. However, in its current design, our protocol may already resist most of these network-level active attacks, because pool-based message re-ordering, dummy messages, controlled traffic rates, and of URE, are known to participate in preventing them (in particular, URE changes messages' appearance in a non-deterministic way, invalidating a base assumption of several mixnets-related attacks).

To offer protection against malicious adversaries, the protocol should also be modified to ensure integrity and authenticity of messages. This can be done on a link-basis, by including Messages Authentication Codes (MAC) [MOV96] in the message headers, or

on an end-to-end basis, *e.g.* by encrypting oriented communication payloads using the AES block cipher in an authenticated encryption mode such as the *Gallois Counter Mode* (GCM) [MV04]. Also, the unauthenticated DHKA used in the protocol should be replaced with one of its many well-studied *secure* variants (*e.g.* using certified public keys). These variants, however, often assume the presence of a trusted entity that issues or certifies the keys, an assumption we carefully avoided so far in the design of the protocol

Finally, there are also attacks specific to our protocol. In particular, during topology dissemination a malicious corrupted node can easily mount several attacks. For instance, by the way route proposals work, when an end-receiver receives a `rtproprelay` message, it automatically decrypts and hashes the ciphertext corresponding to the pseudonym. This offers the opportunity to a malicious corrupted node to embed a ciphertext on which it wants to obtain information into a `rtproprelay` message, and contact the end-receiver. Although this malicious node does not directly obtain the corresponding plaintexts (but a hashed version of it), it obtains partial information on them, thus defeating the purpose of encryption. One way to counter the above attacks is by *forcing the honest behavior* of nodes. Indeed, it is known that that there exist a generic transformation, that transforms any SMPC protocol secure against passive adversaries into one secure against malicious adversaries [Gol04]. In these regards, it is possible to consider the entire topology dissemination phase as one large SMPC, and apply this generic transformation. This would however be very costly, both in terms of computation and bandwidth, because this SMPC transformation relies on a heavy use of *Zero-Knowledge Proofs* (ZKP) and *commitment schemes* [Gol04].

Further Increase the Level of Privacy

The privacy level we aim for in this thesis is stronger than in most existing works, but it is not the maximal anonymity that can be imagined. Indeed, there exist protocols that aim at preventing the detection (by external adversaries) of the very fact that nodes *participate in the anonymous network*. These protocols ensure such a property through the use of *steganography*. For instance, Invernizzi *et al.* [IKV13] propose to hide data in blog posts, and the images they contain. Another approach is to make the traffic of the anonymous protocol *look like* another innocuous protocol, such as HTTP (thus realising a form of *protocol steganography*) [Wei+12]. However, the first method only works for small, unidirectional, and very high latency communications; and Houmansadr *et al.* have shown that the second method is ineffective against simple traffic analysis attacks [HBS13]. Another kind of solution was proposed by Clarke *et al.* in an extension of the Freenet protocol [Cla+10]. The authors propose to hide participation of a node in the network by restricting the neighbors of a node to trusted peers. Since only the neighbors of a node X may see X's IP address, and since those neighbors are trusted by X, a form of participation hiding is achieved (but a *weak* form, since the attacks of Houmansadr *et al.* can still be mounted on links between neighbors). This approach is easily applicable to our protocol, since we assume a connected but incomplete topology graph, similarly to Clarke *et al.* Note however that it is not applicable to a client-server architecture, in which simply observing who connects to the publicly known relay servers

allows to uncover the identity of network users.

Towards More Advanced Routing Functionalities

In this thesis, we mainly focused on providing privacy, to the expense of efficiency. Also, we left aside considerations about network and route management, such as the handling of node failure, the destruction (and reconstruction) of routes, and the management of bandwidth and congestion.

Some of these elements (management of bandwidth in particular) can be taken into account in the route proposal policy. That is, in the same way as the hop count, the overall route bandwidth can be encrypted and advertised in route proposals, allowing the proposee to leverage the homomorphic properties of the Elgamal scheme to determine whether they accept the route or not (without actually learning the route's bandwidth). Other elements, such as management of congestions, can be left as the task of upper network layers. Then, note that the current design of the protocol already partially handles node failure, since nodes have multiple routes towards a same receiver. If one breaks, they still can use the others. The protocol however lacks a way for a node to signal a broken link. Indeed, when the link of a route is broken (*e.g.* because a node left the network), all upstream nodes of the route must be made aware of this event, and must stop using the route. Alternatively, the protocol could be designed with a *route-repair* mechanism which re-attaches the broken route to a valid one, preventing the need to reconstruct the entire route.

In our work, we choose to first design the protocol without all these network components. Indeed the integration of these mechanism may actually degrade privacy, because they can act as side-channels for the adversary. For instance, the adversary could deliberately destroy a route link to see the consequences in the network and how route failure propagates, in order to ultimately try to deduce the identity of nodes on the route.

Opening and Perspectives

The scope of the work presented in this thesis is limited to direct messaging over the Internet. But some elements of the proposed protocol may have other applications, *e.g.* in anonymous file sharing networks. In particular, the use of relationship pseudonyms, the computation of information on routes using homomorphic encryption may be adapted to other architecture. It would also be interesting to see if the same mechanisms can be put in place in a protocol that builds circuits in the more traditional manner (in the manner of *e.g.* Tor), and if the same level of privacy as in our protocol can be achieved. Furthermore, we have presented our protocol as an Internet overlay, for it is the most popular application for anonymous networks. However, our protocol can, at least theoretically, be instantiated over any other network working on a connected-but-incomplete topology graph. This includes mesh wired or wireless networks. The main impediment to such an adaptation is the fact that in mesh networks, in particular wireless ones, nodes are often small devices with low power, that may not be able to support the heavy operations required to run the protocol.

The work we have carried out in this thesis fits into the current debates that has been emerging in Western societies around the freedom of the Internet and the privacy of its users. Our protocol participates in making the Internet a collaborative platform, free of pervasive surveillance, and respectful of the freedom of speech. Although it is not possible to technically ensure that the anonymity provided by our protocol will be used for the public good and not for criminal activities (a common ethical issue in science), we believe that mass surveillance of citizens (even those who have *nothing to hide*) boils down to abandoning the *presumption of innocence*, and is not the solution [Gre14]. Indeed, as explained by Edward Snowden and shown by the recent CIA leaks [Wik17], intelligence agencies do not *need* mass surveillance to catch criminals: they "find ways around [encryption]" [Lee14] when the need arises, on a case-by-case basis.

A. Instantiation of $\mathbb{G}$ and Group Encoding

The Elgamal PKE scheme, the DHKA, and the URE scheme used in this thesis have in common that they function within the same group $\mathbb{G}$. The security of these cryptographic tools (and of the whole protocol) rely on the assumption that the DDH problem is hard in the group $\mathbb{G}$. The choice of group is thus of central importance. This appendix explains in details how the group in instantiated. It also discusses the plaintext *encoding* issue specific to the Elgamal scheme.

A.1. Instantiating the Group

The DDH problem is believed intractable in various groups [Bon98]. The most suitable way to instantiate $\mathbb{G}$ for this thesis, is to take a subgroup of prime order q of the group $(\mathbb{Z}_p^*, \times \bmod p)$ where p is a *safe prime*, *i.e.* $p = 2q+1$. The resulting $\mathbb{G}$ is cyclic and abelian. The primes p and q must be chosen large enough, so that computing discrete logarithms in $\mathbb{G}$ (and solving the DDH problem) is hard enough w.r.t. the security parameter λ. For $\lambda = 128$, it is advised to set $|p| \approx 2048$ and $|q| \approx 200$ [Gir15].

This group can equivalently be described as the group of *quadratic residues* modulo p, meaning that $\mathbb{G} = \left\{ e \mid \exists\, x \in \mathbb{Z}_p^* \text{ s.t. } x^2 = e \bmod p \right\}$. But the simplest way to express it is with a suitable generator g: $\mathbb{G} = \langle g \rangle = \{ g^i \mod p \mid i \in \mathbb{Z}_q \}$, which clearly is composed of q elements. Note that each and every element in $\mathbb{G}$ is a candidate generator of $\mathbb{G}$ (except the identity element). Thus, to find one, it is sufficient to take any element $x \in \mathbb{Z}_p^*$, and if $y = x^2 \bmod p \neq 1$, use y as generator g (since $x^2 \bmod p$ is a quadratic residue modulo p, it has to be an element of $\mathbb{G}$).

In the protocol, all nodes must agree on the same group and the same generator. Note that the group is fully defined by the pair (q, g). These parameters are thus assumed publicly known by all nodes. In practice, they are chosen once and for all, by consensus at the start of the network or by hard-coding them into nodes for instance.

A.2. Encoding of Elgamal Plaintexts

While the issue of plaintext encoding in the Elgamal scheme is known [CPP06], most works often fail to specify how they work around it. However, the matter is crucial: in

the scheme, not everything can be securely encrypted *and* yield ciphertexts that support homomorphic operations.

Historically, Tahrer Elgamal proposed to set the plaintext space of the scheme to to $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$. It is indeed standard to choose a plaintext space that is identifiable to a set $\{1, \ldots, n\}$ (for some $n \in \mathbb{N}$), because then any bit-string of length lower than $|n|$ bits can be directly encrypted. However, this choice turned out to lead to an insecure scheme: the group $\mathbb{Z}_p^*$ is now known to be vulnerable against attacks on the DDH problem [Bon98]. To be secure, the Elgamal scheme *must* encrypt elements from a group $\mathbb{G}$ where the DDH assumption holds, such as $\mathbb{G} = \{g^i \mod p \mid i \in \mathbb{Z}_q, p = 2q+1\}$. As a result, one practical way to implement the Elgamal scheme on the later group is to take a plaintext in $\{1, \ldots, q\}$, and *encode* it into an element of $\mathbb{G}$ before encrypting it. Another common alternative is to use a hash-based encryption algorithm [CPP06]. Both solution however degrade or completely take away the homomorphic properties of the scheme. These limitations and questions about encoding are explored in details by Chevallier-Mames *et al.* [CPP06], and by Marc Joye [Joy16].

In definitive, there are two options: allow the encryption of any $|q|$-bit-string (*i.e.* set the plaintext space to $\{1, \ldots, q\}$, use encoding, and loose homomorphic properties); or directly encrypt elements of $\mathbb{G}$ (*i.e.* have homomorphic properties, but loose the ability to encrypt arbitrary application data).

In this thesis, a hybrid approach is chosen. Indeed, only ciphertexts from *routing messages* need to support homomorphic operations. Thus, all terms and plaintext used in routing messages are taken directly in $\mathbb{G}$. On the other hand, payload messages' plaintexts consists in arbitrary application data, but do not need to support homomorphic operations. They can therefore be encoded from $\{1, \ldots, q\}$ to $\mathbb{G}$. In the description of the protocol, the encoding of payload message is implicit and ignored. For completeness however, note that for the present choice of group, $m \in \{1, \ldots, q\}$ can be encoded as $e = m^2 \mod p$, which can efficiently be inverted by computing $e^{1/2 \bmod \phi(p)} \mod p$ using Euler's theorem [Sho09; Joy16].

B. Detailed Cryptographic Proofs

This appendix presents the full proofs of all theorems from Chapter 5, in their order of appearance. For more context, the reader is invited to refer to the section of chapter where the theorem is formulated. The last section is however an attempt at UC-realising the ideal functionality $\mathcal{F}_{\mathsf{rtprop}}$. Because the UC proofs sand in the $\mathcal{F}_{\mathsf{rtprop}}$-hybrid model, the idea is to break down the assumption that this ideal functionality represents.

B.1. Proof of Theorem 1 (Pseudonyms Security)

This proof relates to the three properties of pseudonyms defined in Section 5.4 (p. 104).

Proof of Theorem 1. Each property is proven independently.

– *Proof of Uniqueness.* Since the hash function h is collision-resistant, then for any fixed $src \in \mathbb{Z}_q^*$:

$$\begin{aligned}
&\Pr[dst_1 \neq dst_2 \wedge f(src, dst_1) = f(src, dst_2) \mid dst_1, dst_2 \leftarrow_{\$} \mathbb{G}] \\
=&\Pr[x \neq x' \wedge \mathsf{h}(x) = \mathsf{h}(x') \mid dst_1, dst_2 \leftarrow_{\$} \mathbb{G}; x \coloneqq dst_1^{src}; x' \coloneqq dst_2^{src}] \\
\leq&\mathsf{Adv}_{\mathcal{A}}^{\mathsf{h}-\mathsf{coll}}(\lambda)
\end{aligned}$$

Additionally, it can happen that dst_1 and dst_2, drawn at random from $\mathbb{G}$, are equal. This happens with probability $1/|\mathbb{G}|$. Consequently, it holds that:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{ps-uniq}}_{\mathcal{A}}(\lambda) &= \Pr[f(src, dst_1) = f(src, dst_2) \mid dst_1, dst_2 \leftarrow_{\$} \mathbb{G}] \\
&= \Pr[f(src, dst_1) = f(src, dst_2) \mid dst_1, dst_2 \leftarrow_{\$} \mathbb{G}; dst_1 = dst_2] \\
&\qquad \cdot \Pr[dst_1 = dst_2 \mid dst_1, dst_2 \leftarrow_{\$} \mathbb{G}] \\
&\quad + \Pr[f(src, dst_1) = f(src, dst_2) \mid dst_1, dst_2 \leftarrow_{\$} \mathbb{G}; dst_1 \neq dst_2] \\
&\qquad \cdot \Pr[dst_1 \neq dst_2 \mid dst_1, dst_2 \leftarrow_{\$} \mathbb{G}] \\
&= 1 \cdot \Pr[dst_1 = dst_2 \mid dst_1, dst_2 \leftarrow_{\$} \mathbb{G}] \\
&\quad + \Pr[dst_1 \neq dst_2 \wedge f(src, dst_1) = f(src, dst_2) \mid dst_1, dst_2 \leftarrow_{\$} \mathbb{G}] \\
&= \frac{1}{|\mathbb{G}|} + \mathsf{Adv}^{\mathsf{h-coll}}_{\mathcal{A}}(\lambda)
\end{aligned}$$

Since $\mathsf{Adv}^{\mathsf{h-coll}}_{\mathcal{A}}(\lambda)$ is assumed negligible in λ, and $|\mathbb{G}| = q \approx 2^{\mathcal{O}(\lambda)}$, $\mathsf{Adv}^{\mathsf{ps-uniq}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$. □

– *Proof of one-wayness .* If there exists an adversary $\mathcal{A}$ successfully outputting $dst_{\mathcal{A}}$ such that $f(src, dst_{\mathcal{A}}) = f(k, src, dst)$, then it is possible to construct an adversary $\mathcal{B}$ that distinguishes h from a random function $\mathtt{Rand}$. Denote by $\mathcal{O}$ the challenge function given to $\mathcal{B}$. That is, either $\mathcal{O} = \mathsf{h}$, or $\mathcal{O} = \mathtt{Rand}$. $\mathcal{B}$ has oracle access to $\mathcal{O}$, but it is important to note that, in any case, $\mathcal{A}$ *also* has free access to it. That is, $\mathcal{O}$ is public, and $\mathcal{A}$ can compute pseudonyms for any src and dst values that it wants (just as it would do with a regular hash function).

$\mathcal{B}$ is constructed as follows, for any given src value. It starts by sampling $dst \leftarrow_{\$} \mathbb{G}$, and giving src to $\mathcal{A}$. Then, whenever $\mathcal{A}$ makes a request to its oracle $f(\cdot, dst)$ with value src, then $\mathcal{B}$ calls $\mathcal{O}$ on the value dst^{src} and sends the result back to $\mathcal{A}$. Ultimately, $\mathcal{A}$ outputs $dst_{\mathcal{A}}$. The adversary $\mathcal{B}$ checks if $\mathcal{O}(dst^{src}) = \mathcal{O}({dst_{\mathcal{A}}}^{src})$ by calling its oracle again. If it is the case, $\mathcal{B}$ returns 0 (meaning, $\mathcal{B}$ believes it is interacting with h), and 1 otherwise. When $\mathcal{A}$ succeeds with advantage $\mathsf{Adv}^{\mathsf{ps-ow}}_{\mathcal{A}}(\lambda)$, then $\mathcal{B}$ has advantage $\mathsf{Adv}^{\mathsf{ps-ow}}_{\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{h}/\mathtt{Rand}}_{\mathcal{B}}(\lambda) + 1/|\mathbb{G}| + 1/|\{0,1\}^n|$. Indeed, the probability that $\mathcal{A}$ wins when $\mathcal{B}$ has access to h is by definition $\mathsf{Adv}^{\mathsf{ps-ow}}_{\mathcal{A}}(\lambda)$. And the probability that $\mathcal{A}$ wins when $\mathcal{B}$ has access to a $\mathtt{Rand}$ is zero (since $\mathcal{A}$ then has no information at all on dst), added to the trivial probability that $\mathcal{A}$ randomly guesses dst plus the probability that a collision occurs in the $\mathtt{Rand}$ function. Since both both $\mathbb{G}$ and $\{0,1\}^n$ have size in $2^{\mathcal{O}(\lambda)}$ and $\mathsf{Adv}^{\mathsf{h}/\mathtt{Rand}}_{\mathcal{B}}(\lambda)$ is negligible by assumption, we obtain the proof of one-wayness. □

– *Proof of Indistinguishability .* If there exists an adversary $\mathcal{A}$ breaking the indistinguishability property, then it is possible to construct an adversary $\mathcal{B}$ that distinguishes the hash function from a random oracle. The adversary $\mathcal{B}$ is constructed as follows. At first, it samples $dst_0, dst_1 \leftarrow_{\$} \mathbb{G}$ and $b \leftarrow_{\$} \{0,1\}$, computes

$PS_1 \coloneqq \mathcal{O}(dst_b^{src})$ and $PS_2 \coloneqq \mathcal{O}(dst_{1-b}^{src})$, and hands them to $\mathcal{A}$. Then, for every oracle call with src', $\mathcal{B}$ answers with $\mathcal{O}(dst_0^{src'})$ or $\mathcal{O}(dst_1^{src'})$ (depending on which oracle $\mathcal{A}$ queried). Ultimately, $\mathcal{A}$ outputs b^*. If $b^* = b$, then $\mathcal{B}$ outputs 0 (meaning $\mathcal{B}$ thinks $\mathcal{O} = \mathsf{h}$), and 1 otherwise.

Clearly, $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{h}/\mathtt{Rand}}(\lambda) = \mathsf{Adv}_{\mathcal{A}}^{\mathsf{ps-ind}}(\lambda)$, since when $\mathcal{O} = \mathsf{h}$, $\mathcal{B}$ perfectly emulates $\mathcal{A}$'s challenger, and when $\mathcal{O} = \mathtt{Rand}$, $\mathcal{A}$ can only win win probability $1/2$. Therefore, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ps-ind}}(\lambda) \leq \mathsf{Adv}_{\mathcal{B}}^{\mathsf{h}/\mathtt{Rand}}(\lambda) \leq \mathsf{negl}(\lambda)$, which is negligible by assumption. □

This concludes the proof of Theorem 1. □

B.2. Adversary Model and Notations in the UC Framework

This section details how our adversary model (passive static) is modeled into the UC framework. We present this section before the full proofs in order to avoid any confusion about the formalism and models we use, since, indeed, the flexibility of the UC framework allows to model a same type of adversary in several ways. In this thesis, we use the same approach as Canetti *et al.* in a paper from 2002 [Can+02], which shows proofs for passive static adversaries.

B.2.1. $\mathcal{A}$ as a Proxy vs. $\mathcal{A}$ as an Algorithm

In the UC framework, there are two (equivalent) ways to consider the adversary. To understand those two ways, it is first necessary to understand that the *actual* adversary is really the environment $\mathcal{E}$: it is the one that, in the proofs, must be shown unable to distinguish between the ideal and real executions. In the UC framework, $\mathcal{E}$ is however not allowed to send messages directly to the (ITIs of the) parties of the protocol in the real execution, nor to interact with the ideal functionality in the ideal execution. The environment is only allowed to have an impact on the protocol execution, through the adversary $\mathcal{A}$ in a real execution. In this sense, $\mathcal{A}$ is essentially a proxy of $\mathcal{E}$, allowed to interact with ITIs of nodes. Note that, although this separation into $\mathcal{E}$ and $\mathcal{A}$ seems to be equivalent to allowing $\mathcal{E}$ to directly interact with parties, it has crucial theoretical implications on the way proofs are carried out, and on their impact. Indeed, it is this difference with the *standard model* which allows to prove protocols secure under *universal, concurrent composition*.

Coming back to our main point, there are two ways to see this proxy $\mathcal{A}$: either it is really just an interface, which merely executes the instructions of $\mathcal{E}$; either $\mathcal{A}$ is an algorithm with its own strategy (but a strategy bounded to the environment's will) [Can13]. These two ways are actually equivalent, since $\mathcal{A}$ and $\mathcal{E}$ are not restricted in the information they can exchange. In particular, $\mathcal{A}$ can at any moment send its full state to $\mathcal{E}$, and the latter can "set the code" of $\mathcal{A}$. These two possible ways of considering $\mathcal{A}$ however have an impact on the proof setup, and, in this thesis, in the *notations*. If $\mathcal{A}$ is assumed to be an algorithm on its own, in the ideal execution, $\mathcal{S}\mathtt{im}$ will be set to run an internal

copy of $\mathcal{A}$, and relay the communications between $\mathcal{E}$ and $\mathcal{A}$. If $\mathcal{A}$ is considered to be a proxy, $\mathcal{S}\mathtt{im}$ will not actually run a copy of $\mathcal{A}$, but simply execute (or rather, *simulate the execution* of) the instructions from $\mathcal{E}$. But the two cases are actually equivalent, *as long as*, in the former approach, $\mathcal{S}\mathtt{im}$ relays the communication between $\mathcal{E}$ and its internally ran instance of $\mathcal{A}$ without tampering in any way with the communicated data.

In this thesis, similarly to Canetti *et al.* [Can+02], we choose to consider $\mathcal{A}$ as an algorithm on its own. The reason why we set out to recall that technicality of the UC framework is to avoid confusion. Indeed, we now introduce the notations $\mathcal{A}(X)$ and $\mathcal{S}\mathtt{im}(X)$, used in the proofs contained in this appendix, in order to make explicit the difference between a corrupted node played by $\mathcal{A}$ and an honest node simulated by $\mathcal{S}\mathtt{im}$. Also, in proofs, we write "$\mathcal{E}$ passes input I to $\mathcal{A}(X)$" as a shorthand for "$\mathcal{E}$ communicates input I, meant for corrupted node X to $\mathcal{S}\mathtt{im}$, which directly passes it to its internal copy of $\mathcal{A}$ ". Independently, we also denote the dummy party corresponding to node X by $\mathcal{F}_{\mathsf{rtprop}}(X)$. These notations put in evidence the fact that, in a UC proof, each node X has two *representatives*: X implicitly appears in $\mathcal{F}_{\mathsf{rtprop}}$, and it is also either simulated by $\mathcal{S}\mathtt{im}$ (if honest), or played by $\mathcal{A}$ (if corrupted).

B.2.2. The Passive Static Adversary Model in the UC Framework

In the UC framework, party corruption is not modeled by default. It must be explicitly *coded* into ideal functionalities in particular, using special *corruption* messages. Canetti proposes several ways to implement these messages, for different adversary models [Can13]. In this thesis, to model static passive adversaries, we follow the approach of Canetti *et al.* in a 2002 paper [Can+02].

In the static adversary model, no *corruption* messages are actually used, but it is assumed that prior to the start of the (real or ideal) execution, the adversary specifies all the parties she wants to corrupt. Of course, $\mathcal{E}$ is aware of these corruptions. Furthermore, in the ideal execution, the ideal functionality is also made aware of these corruptions. Then, when a party is corrupted in the real execution, it is $\mathcal{A}$ which takes on its role. That is, $\mathcal{A}$ receives its inputs from $\mathcal{E}$, and sends the messages on behalf of the party. However, in the passive adversary model, it is considered that, $\mathcal{A}$ strictly follows the code of the protocol[1].

In consequence, we consider that, in the real execution, $\mathcal{E}$ does not give inputs to the corrupted parties, but instead directly hand them to $\mathcal{A}$. Similarly, in the ideal execution, $\mathcal{E}$ does not provide inputs to dummy parties of corrupted nodes, but to $\mathcal{S}\mathtt{im}$ (which relays them to $\mathcal{A}$). Finally, note that, because we consider static corruptions, the ideal functionalities presented in Chapter 5 do not need to handle *corruption* messages.

[1] An alternative modeling consists in letting the party continue its execution, but let the adversary have read access to its internal state at any time.

B.3. Proof of Theorem 2 (Π_{rtprop} UC-realises $\mathcal{F}_{\mathsf{rtprop}}$)

The below proof shows that the route proposal protocol Π_{rtprop} (defined p. 108) UC-realises $\mathcal{F}_{\mathsf{rtprop}}$ (defined p. 110). Based on the remarks from the previous section, we denote corrupted nodes played by adv as $\mathcal{A}(X)$, the honest nodes simulated by $\mathcal{S}\mathtt{im}$ as $\mathcal{S}\mathtt{im}(X)$, and the dummy parties as $\mathcal{F}_{\mathsf{rtprop}}(X)$.

Proof of Theorem 2. Let $\mathcal{A}$ be a semi-honest, static adversary that interacts with parties running the protocol Π_{rtprop} in the $(\mathcal{F}_{\mathrm{reg}}, \mathcal{F}_{\mathsf{link}})$-hybrid model. We construct a simulator $\mathcal{S}\mathtt{im}$ for the ideal process for $\mathcal{F}_{\mathsf{rtprop}}$ running in the $\mathcal{F}_{\mathrm{reg}}$-hybrid model such that no environment $\mathcal{E}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and the protocol, or with $\mathcal{S}\mathtt{im}$ and the ideal functionality. Figure 5.7 in Section 5.5 (p. 112) shows the setup and relations between ITIs to carry out the proof. As usual, the simulator $\mathcal{S}\mathtt{im}$ starts by invoking a copy of $\mathcal{A}$. $\mathcal{S}\mathtt{im}$ also runs a simulated version of $\mathcal{F}_{\mathsf{link}}$. However, $\mathcal{F}_{\mathrm{reg}}$ stays outside of $\mathcal{S}\mathtt{im}$.

We describe how $\mathcal{S}\mathtt{im}$ simulates a real execution, by acting on behalf of the honest nodes that $\mathcal{A}$ interacts with, and show that this simulation is indistinguishable from a real execution for the $\mathcal{A}$ and $\mathcal{E}$. For that, the proof is divided in three main parts. The first part gives details on the proof setup; while the second (resp. third) part deals with route proposals towards honest (resp. corrupted) end-receivers. The two latter parts are organised as a *scenario*, divided in several simulation *cases*. In the first simulation case, the honest (resp. corrupted) end-receiver self-proposes; in the second one, that proposition is relayed *etc.*. This ensures no simulation case is forgotten. Each simulation case in the scenario refer to the appropriate modeled leakage formalised by $\mathcal{F}_{\mathsf{rtprop}}$ in Fig. 5.6 (p. 110), using roman numerals.

Simulating the communication between $\mathcal{E}$ and $\mathcal{A}$: $\mathcal{S}\mathtt{im}$ internally runs a copy of $\mathcal{A}$. Every input from $\mathcal{E}$ is written on $\mathcal{A}$'s input tape, and conversely. Note that seeing setup inputs of given by $\mathcal{E}$ to each corrupted node $\mathcal{A}(X)$ allows $\mathcal{S}\mathtt{im}$ to learn the *src* and *dst* value of all corrupted nodes.

Simulating $\mathcal{F}_{\mathsf{link}}$: $\mathcal{S}\mathtt{im}$ internally simulates $\mathcal{F}_{\mathsf{link}}$ completely honestly.

Simulating key pairs: $\mathcal{S}\mathtt{im}$ must obtain a key pair for each honest node, but can not request $\mathcal{F}_{\mathrm{reg}}$ for it, since, for an honest node X, $\mathcal{F}_{\mathrm{reg}}$ only answers to the node itself. Consequently, $\mathcal{S}\mathtt{im}$ generates one fresh key pair for each honest node. Denote the key of $X \in \Omega_{\mathcal{S}\mathtt{im}}$ as $(pk_{\mathcal{S}\mathtt{im}(X)}, sk_{\mathcal{S}\mathtt{im}(X)})$. Additionally, $\mathcal{S}\mathtt{im}$ honestly relays $\mathcal{A}$'s queries to $\mathcal{F}_{\mathrm{reg}}$ for the key pairs of corrupted nodes, and the corresponding answers. Although $\mathcal{S}\mathtt{im}$ does see the key pairs of corrupted nodes in this process, it does not need this knowledge to perform the simulation.

Handling corrupted parties' output/inputs: $\mathcal{S}\mathtt{im}$ gives the corrupted parties' inputs to $\mathcal{S}\mathtt{im}$ which, as mentioned above, passes them directly to $\mathcal{A}$. However, in addition to giving them to $\mathcal{A}$, $\mathcal{S}\mathtt{im}$ gives a copy of all setup inputs given by $\mathcal{E}$ to $\mathcal{F}_{\mathsf{rtprop}}$. Other inputs are handled case by case, as described in subsequent paragraphs. In any case, $\mathcal{S}\mathtt{im}$ always receives the $\mathsf{proposee}$ outputs of $\mathcal{F}_{\mathsf{rtprop}}(X)$

for all corrupted X. But $\mathcal{S}\mathtt{im}$ does not forward them to anyone: $\mathcal{A}$ will be making its own $\mathsf{proposee}$ outputs, that $\mathcal{S}\mathtt{im}$ will then pass on to $\mathcal{E}$.

Simulating intermediary corrupted sub-paths (abstractly): $\mathcal{S}\mathtt{im}$ can receive, at any moment, an *intermediary corrupted sub-path* leak $(\mathsf{subpath}, sid, (Z_{i'} \xrightarrow{cid_{i'+1}} \dots \xrightarrow{cid_{j'}} Z_{j'}))$ from $\mathcal{F}_{\mathsf{rtprop}}$. $\mathcal{S}\mathtt{im}$ can not (in the general case) link it to a particular route proposal. This is not an issue, since each such sub-path can be simulated independently. However, it is crucial that sub-paths be simulated in the exact order in which they are leaked by $\mathcal{F}_{\mathsf{rtprop}}$.

For clarity, the formal description of sub-paths simulation is deferred to the end of the proof. In a nutshell, the idea is for $\mathcal{S}\mathtt{im}(Z_{i'})$ (*i.e.* $\mathcal{S}\mathtt{im}$, simulating honest node $Z_{i'}$) to create a message with adequate ciphertexts (in particular encrypted under the appropriate public keys), and to send it to $\mathcal{A}(Z_{i'+1})$ (*i.e.* to $\mathcal{A}$, that plays the role of corrupted node $Z_{i'+1}$) with $cid_{i'+1}$. By construction of the simulation, and because $\mathcal{A}$ is semi-honest, it is ensured that the message will exactly follow the leaked sub-path, and arrive to $\mathcal{S}\mathtt{im}(Z_{j'})$ with $cid_{j'}$. We will show that this simulation is indistinguishable from a real execution.

First scenario: route proposals towards an honest end-receiver. We now describe the simulation of actual route proposals. Since each route is constructed hop by hop, it is clearer to describe the simulator starting from a self-proposition, and then show how relayed propositions propagate. There are two main scenarios, depending on whether the self-proposing end-receiver is honest or corrupted. We start by an honest end-receiver. The scenario is based on the route described in Fig. B.1, where corrupted nodes are designated with a super script star. That is, the scenario begins with the honest end-receiver R self-proposing to a corrupted Z_1^*, the latter then relaying the route proposal to Z_2^*, and so on.

Figure B.1. – Route for the First Scenario, with a Honest R

$\mathcal{S}\mathtt{im}(R)$ ***self-proposes*** (V)***:*** First of all, the honest node R self-proposes. That means that $\mathcal{F}_{\mathsf{rtprop}}$, receives $(\mathsf{proposer}, sid, (Z_1^*, cid_{Z_1^*\text{-}R}), PS_{R\to R}, \mathtt{null})$ from the dummy party $\mathcal{F}_{\mathsf{rtprop}}(R)$. It is important to note that, since R is honest, $\mathcal{E}$ provides the above input to $\mathcal{F}_{\mathsf{rtprop}}(R)$. Thus, $\mathcal{S}\mathtt{im}$ does not get to see that input, and does not immediately knows that it must simulate a self proposition by $\mathcal{S}\mathtt{im}(R)$ to $\mathcal{A}(Z_1^*)$. However, $\mathcal{S}\mathtt{im}$ may will leaks from $\mathcal{F}_{\mathsf{rtprop}}$ relating to this self-proposal: $\mathcal{F}_{\mathsf{rtprop}}$ leaks $(\mathsf{rtprop}, sid, Z_1^* \xleftrightarrow{cid_{Z_1^*\text{-}R}} R \to ?)$.

$\mathcal{S}\mathtt{im}$ proceeds as follows, upon receiving such a leak. First, $\mathcal{S}\mathtt{im}(R)$ samples $r \leftarrow_{\$} \mathbb{G}$, and sends to $\mathcal{A}(Z_1^*)$:

$$\left\langle \mathtt{rtprop} \| cid_{Z_1^*\text{-}R}, \mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(R)}, r), \mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(R)}, 1) \right\rangle$$

Consequently, when $\mathcal{A}(Z_1^*)$ answers with $\left\langle \mathtt{rtprop} \| cid_{Z_1^*\text{-}R}, c_1, c_2 \right\rangle$, $\mathcal{S}\mathtt{im}(R)$ decrypts c_2 to get pk^{tmp}. At some point, $\mathcal{F}_{\mathsf{rtprop}}(Z_1^*)$ will output $(\mathsf{proposee}, sid, PS_{Z_1^*\to R}, c_{\mathtt{one}Z_1^*\to R}, (R, cid))$ to $\mathcal{S}\mathtt{im}$, allowing $\mathcal{S}\mathtt{im}$ to learn $PS_{Z_1^*\to R}$. Then, and only then, $\mathcal{S}\mathtt{im}(R)$ sends the final message

$$\left\langle \mathtt{rtprop} \| cid_{Z_1^*\text{-}R}, \mathsf{Enc}(pk^{tmp}, PS_{Z_1^*\to R}), \mathsf{Enc}(pk^{tmp}, 1) \right\rangle$$

to $\mathcal{A}(Z_1^*)$. Even though the first ciphertext contains r instead of dst_R, and is encrypted under the *wrong* public key, this simulation is indistinguishable from a real execution, by the IND-CPA and IK-CPA properties. More formally, the proof requires a game-hop: the first game requires to distinguish between $\mathsf{Enc}(pk_R, dst_R)$ and $\mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(R)}, r)$ (which corresponds to distinguishing the real and simulated executions); the second game requires to distinguish between $\mathsf{Enc}(pk_R, dst_R)$ and $\mathsf{Enc}(pk_R, r)$; and the last game requires to distinguish between $\mathsf{Enc}(pk_R, r)$ and $\mathsf{Enc}(k_R, r)$. Clearly, the advantage of the adversary is zero in the last game. It is possible to show that if there exists a distinguisher between the two first games, then there exists an adversary $\mathcal{B}$ that breaks IK-CPA (the idea being that $\mathcal{B}$ ties its challenge public keys to pk_R and $pk_{\mathcal{S}\mathtt{im}(R)}$; and if there exists a distinguisher between the two last games, then there exists an adversary $\mathcal{B}$ breaking IND-CPA (the idea is for $\mathcal{B}$ to issue $(m_0, m_1) \coloneqq (dst_R, r)$ as challenge).

$\mathcal{A}(Z_1^*)$ proposes the route to $\mathcal{A}(Z_2^*)$ (VI): In a second step, $\mathcal{A}(Z_1^*)$ then proposes the route it just learned to another corrupted node $\mathcal{A}(Z_2^*)$. That is, $\mathcal{E}$ gives a $(\mathsf{proposer}, sid, (Z_2^*, cid_{Z_2^*\text{-}Z_1^*}), PS_{Z_1^*\to R}, (R, cid_{Z_1^*R}))$ input to $\mathcal{A}(Z_1^*)$. $\mathcal{S}\mathtt{im}$ sees that input as it relays it from $\mathcal{E}$ to $\mathcal{A}$, and recognises $PS_{Z_1^*\to R}$ and $(R, cid_{Z_1^*\text{-}R})$ as the route that $\mathcal{S}\mathtt{im}(R)$ previously proposed. $\mathcal{A}$ will begin by playing the protocol for Z_1^* and Z_2^*, using $\mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(R),Z_1^*}, r)$ as $c_{\mathtt{prop}}$, since it corresponds to the one it learned from $\mathcal{S}\mathtt{im}(R)$ earlier. At some point, $\mathcal{S}\mathtt{im}(R)$ necessarily receives from $\mathcal{A}(Z_1^*)$

$$\left\langle \mathtt{rtproprelay} \| cid \| rcid, c_1 = \mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(R)}, r^{src_{Z_2^*}}), c_2 = \mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(R)}, pk^{tmp}) \right\rangle$$

At that point, $\mathcal{S}\mathtt{im}(R)$ can decrypt c_1 and c_2 to get pk^{tmp} and $r^{src_{Z_2^*}}$ which it knows to be related to Z_2^* (since it knows r and $src_{Z_2^*}$). At this point, $\mathcal{S}\mathtt{im}$ externally activates $\mathcal{F}_{\mathsf{rtprop}}(Z_1^*)$ with a copy of the $\mathsf{proposer}$ input. Consequently, $\mathcal{S}\mathtt{im}$ receives $\mathcal{F}_{\mathsf{rtprop}}(Z_2^*)$'s output $(\mathsf{proposee}, sid, PS_{Z_2^*\to R}, c_{\mathtt{one}Z_2^*\to R}, (Z_1^*, cid_{Z_2^*\text{-}Z_1^*}))$. The simulator $\mathcal{S}\mathtt{im}$ now knows the pseudonym expected by $\mathcal{A}(Z_2^*)$, and $\mathcal{S}\mathtt{im}(R)$ can send it back in a `rtproprelay` message, using the same $rcid$ as in the received one.

The simulation is thus similar to the case of R self-proposing, and indistinguishability is proven with the same arguments. Note that when $\mathcal{A}(Z_1^*)$ simultaneously proposes the route to two (or more) corrupted nodes $\mathcal{A}(Z_2^*)$ and $\mathcal{A}(Z_2^{*\prime})$, the simulation still works. Although the concurrency of the two route proposal seems to prevent $\mathcal{S}\mathtt{im}$ from linking the right $rcid$ value with the right pseudonym $PS_{Z_2^*\to R}$

or $PS_{Z_2^{*\prime}\to R}$ to send back, $\mathcal{S}\mathtt{im}$ can get over this by recognising and distinguishing the values $r^{src_{Z_2^*}}$ and $r^{src_{Z_2^{*\prime}}}$.

$\mathcal{A}(Z_2^*)$ ***proposes the route to*** $\mathcal{A}(Z_3^*)$ ***and so on*** (VI)**:** The above simulation methodology works because $(R, cid_{Z_1^*\text{-}R})$ appears in the proposer input. However, the strategy can be generalized even if $\mathcal{A}(Z_2^*)$ then proposes the route to another corrupted node $\mathcal{A}(Z_3^*)$ and so on. Indeed, $\mathcal{S}\mathtt{im}$ can see each proposer inputs given by $\mathcal{E}$ to the corrupted nodes. These inputs specify the new *cid* of the link to be formed, along with the existing next hop on which the route extends. Since these proposer inputs are given sequentially and in order, as long as the route is relayed among corrupted proposers and proposees, $\mathcal{S}\mathtt{im}$ perfectly knows the full route being built towards $\mathcal{S}\mathtt{im}(R)$. Thus, in the general case, when a corrupted node receives a new proposer input, $\mathcal{S}\mathtt{im}$ can trace the route by following the *cid*s, and *e.g.* check if it goes towards $\mathcal{S}\mathtt{im}(R)$. The concurrency between two or more route proposals also occurs in this case, and is also resolved by the same methodology. Indistinguishability is again proved with the IND-CPA and IK-CPA properties.

$\mathcal{A}(Z_n^*)$ ***proposes the route to*** $\mathcal{S}\mathtt{im}(X)$ (VI)**:** Let's jump ahead in the scenario, when $\mathcal{A}(Z_n^*)$ relays the proposition to the honest node $\mathcal{S}\mathtt{im}(X)$. That is, $\mathcal{A}(Z_n^*)$ is instructed by $\mathcal{E}$ to propose a route to $\mathcal{S}\mathtt{im}(X)$ with a $(\mathsf{proposer}, sid, (X, cid_{X\text{-}Z_n^*}), PS_{Z_n^*\to R}, (Z_{n-1}^*, cid'_{Z_n^*\text{-}Z_{n-1}^*}))$ input. $\mathcal{S}\mathtt{im}$ gets to see that input, can trace the *cid*s links, and guess that this route ends at honest node $\mathcal{S}\mathtt{im}(R)$. When $\mathcal{A}(Z_n^*)$ sends the first `rtprop` message to $\mathcal{S}\mathtt{im}(X)$, with $c_1 = \mathsf{Enc}(pk_{Z_n^*,\ldots,Z_1^*,\mathcal{S}\mathtt{im}(R)}, r)$, $\mathcal{S}\mathtt{im}(X)$ uses some $r' \leftarrow_{\$} \mathbb{Z}_q^*$ to play the role of src_X. For the rest (the ScExp operation, and the key pk^{tmp}), it behaves honestly and answers $\mathcal{A}(Z_n^*)$. Consequently, a `rtproprelay` message is forwarded on the route $\mathcal{A}(Z_n^*), \ldots, \mathcal{A}(Z_1^*)$, and arrives at $\mathcal{S}\mathtt{im}(R)$ under the form

$$\left\langle \mathtt{rtproprelay} \| cid_{Z_1^*\text{-}R} \| rcid, c_1 = \mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(R)}, r^{r'}), c_2 = \mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(R)}, pk^{tmp\prime}) \right\rangle$$

The simulator $\mathcal{S}\mathtt{im}$ decrypts the ciphertexts, thus extracting $pk^{tmp\prime}$ and $r^{r'}$, the latter of which allows to link this message with $\mathcal{S}\mathtt{im}(X)$. $\mathcal{S}\mathtt{im}$ then waits for the $(\mathsf{rtprop}, sid, X \xleftarrow{cid_{X\text{-}Z_n^*}} (Z_n^* \xrightarrow{cid_{Z_n^*\text{-}\cdots}} \ldots \xrightarrow{cid_{Z_1^*\text{-}R}} R))$ leak from $\mathcal{F}_{\mathsf{rtprop}}$. When it is received (and only then), $\mathcal{S}\mathtt{im}(R)$ answers along the reversed route with the same $rcid$, and the same value $r^{r'}$ instead of a real pseudonym. When the final `rtprop` message gets to $\mathcal{S}\mathtt{im}(X)$, the latter decrypts the ciphertexts, recognises $r^{r'}$, and simply discards the ciphertexts. Finally, $\mathcal{S}\mathtt{im}$ must give a copy of the proposer input to $\mathcal{F}_{\mathsf{rtprop}}(Z_n^*)$ so that $\mathcal{F}_{\mathsf{rtprop}}(X)$ makes its proposee output to $\mathcal{E}$.

This simulation is indistinguishable from a real execution, by the IND-CPA and IK-CPA properties, as in previous cases, and by the USS property as well. Indeed, in this simulation case, the $c_{\mathtt{one}X\to R}$ output by $\mathcal{F}_{\mathsf{rtprop}}(X)$ to $\mathcal{E}$ is totally independent of the $c_{\mathtt{one}} = \mathsf{Enc}(pk_{Z_n^*,\ldots,Z_1^*,\mathcal{S}\mathtt{im}(R)}, 1)$ given out by $\mathcal{A}(Z_n^*)$ in its first `rtprop` message. However, if there exists a distinguisher (based on this fact) between the simulation and a real execution, then it is possible to construct an adversary breaking the USS property.

$\mathcal{S}\mathtt{im}(X)$ ***proposes the route*** (V)***:*** Pushing further the scenario, what if $\mathcal{S}\mathtt{im}(X)$ must then re-propose the route? If the proposee to which $\mathcal{S}\mathtt{im}(X)$ proposes the route is honest, there is nothing to do (actually, $\mathcal{S}\mathtt{im}$ will have no information on that route proposal). If the proposee is corrupted, however, we return to the first simulation case. Indeed, for an honest end-receiver, $\mathcal{S}\mathtt{im}$ can not distinguish between a self-proposition and a relayed one. That is, $\mathcal{S}\mathtt{im}$ will not actually be able to *recognise* the route, because it can no longer trace the sequence of *cid*, and what $\mathcal{F}_{\mathsf{rtprop}}$ will leak for this new route proposal will not be linkable by $\mathcal{S}\mathtt{im}$ to the above scenario. Consequently, $\mathcal{S}\mathtt{im}$ acts in the same way whether an honest node is self-proposing or relaying a proposition. Abstractly, this means that $\mathcal{S}\mathtt{im}(X)$ will now play the role that $\mathcal{S}\mathtt{im}(R)$ thus far played in this scenario. Furthermore, $\mathcal{S}\mathtt{im}(R)$ will actually only be solicited as part of what seems like an intermediary corrupted sub-path ($X \overset{cid_{X\text{-}Z^*_n}}{\longrightarrow} Z^*_n \overset{cid_{Z^*_n\text{-}\cdots}}{\longrightarrow} \ldots \overset{cid_{Z^*_1\text{-}R}}{\longrightarrow} R$). There is an important difference however: the return trip on to R (on the said intermediary sub-path) must be simulated *before* $\mathcal{S}\mathtt{im}(X)$ sends the final message of the route proposal. This is done *automatically* since, as briefly mentioned earlier, $\mathcal{S}\mathtt{im}$ will automatically simulate intermediary corrupted sub-paths as they are leaked; and because in the first simulation case, and the one immediately above, note that $\mathcal{S}\mathtt{im}$ respectively waits for a proposee output and a rtprop leak before answering the route proposal. All this ensures the good ordering of events from $\mathcal{A}$'s point of view.

Second scenario: route proposals towards a corrupted end-receiver. This second scenario (which is also the last one), deals with a corrupted end-receiver. The proof for this scenario is outlined similarly to the previous one, starting from R^* self-proposing, and then other nodes relaying this route proposal. The route being built at the outcome of the scenario is depicted in Fig. B.2

Figure B.2. – Route for the Second Scenario, with a Corrupted R

$\mathcal{A}(R^*)$ ***self-proposes to*** $\mathcal{A}(Y^*)$***:*** In this case, $\mathcal{A}(R^*)$ receives an input of the form $(\mathsf{proposer}, sid, (Y^*, cid_{Y^*\text{-}R^*}), PS_{R^*\to R^*}, \mathtt{null})$, which $\mathcal{S}\mathtt{im}$ sees. $\mathcal{S}\mathtt{im}$ can detect that this is a self-proposition from the `null` value of the next hop. Because the proposee Y^* is also corrupted, and thus played by $\mathcal{A}$, there is actually nothing to simulate, but only to feed a copy of the proposer input to $\mathcal{F}_{\mathsf{rtprop}}(R^*)$ so that the internal state of nodes in $\mathcal{F}_{\mathsf{rtprop}}$ matches the simulation. More generally, when the proposer, proposee, end-receiver, and the relay nodes in between are all corrupted, $\mathcal{S}\mathtt{im}$ has nothing to simulate.

$\mathcal{A}(Y^*)$ ***proposes the route to*** $\mathcal{S}\mathtt{im}(Z_1)$ (I)***:*** Now, it is $\mathcal{A}(Y^*)$ that receives an input $(\mathsf{proposer}, sid, (Z_1, cid_{Z_1\text{-}Y^*}), PS_{Y^*\to R^*}, (R^*, cid_{Y^*\text{-}R^*}))$, which $\mathcal{S}\mathtt{im}$ sees. $\mathcal{S}\mathtt{im}$ can detect that this is a relayed proposition towards R^*, in two ways: by the uniqueness of pseudonyms (since, given dst_{R^*} and src_{Y^*}, $\mathcal{S}\mathtt{im}$ can compute $PS_{Y^*\to R^*}$,

which unambiguously designates R^*); and by the next hop $(R^*, cid_{Y^*\text{-}R^*})$, which, as long as there are only corrupted nodes between Y^* and R^*, can be traced back to R^* since $\mathcal{S}\mathtt{im}$ perfectly knows the portions of routes going through corrupted nodes.

Consequently, when $\mathcal{A}(Y^*)$ receives the above proposer input it will start by sending to $\mathcal{S}\mathtt{im}(Z_1)$:

$$\langle \mathtt{rtprop} \| cid_{Z_1\text{-}Y^*}, c_{\text{prop}} \coloneqq \mathsf{Enc}(pk_{Y^*,R^*}, dst_{R^*}), c_{\text{one}} \coloneqq \mathsf{Enc}(pk_{Y^*,R^*}, 1) \rangle$$

To be able to answer, $\mathcal{S}\mathtt{im}$ copies the proposer input to $\mathcal{F}_{\mathsf{rtprop}}(Y^*)$. $\mathcal{S}\mathtt{im}$ then waits to receive the ${dst_{R^*}}^{src_{Z_1}}$ value given in the leak $(\mathsf{rtprop}, sid, Z_1 \xleftrightarrow{cid_{Z_1\text{-}Y^*}} Y^* \xrightarrow{cid_{Y^*\text{-}R^*}} R^*, {dst_{R^*}}^{src_{Z_1}})$. From this, $\mathcal{S}\mathtt{im}(Z_1)$ answers $\mathcal{A}(R^*)$ with ciphertexts $c_1' \leftarrow \mathsf{Enc}_{\mathsf{nopk}}(c_{\text{one}}, {dst_{R^*}}^{src_{Z_1}})$ and $c_2' \leftarrow \mathsf{Enc}_{\mathsf{nopk}}(c_{\text{one}}, pk^{tmp})$ for a newly generated key pk^{tmp}. Ultimately, $\mathcal{A}(R^*)$ answers with ciphertexts encrypted under pk^{tmp}, which $\mathcal{S}\mathtt{im}(Z_1)$ can discard. After this, $\mathcal{S}\mathtt{im}$ sends a continue to $\mathcal{F}_{\mathsf{rtprop}}$, letting $\mathcal{F}_{\mathsf{rtprop}}(Z_1)$ make its proposee output. Finally, $\mathcal{S}\mathtt{im}(Z_1)$ stores c_{one} for later use.

Indistinguishability with a real execution is again obtained from the USS property: even though c_1' and c_2' are not computed by $\mathcal{S}\mathtt{im}(Z_1)$ in the proper way, the re-randomisation performed in the real and simulated executions ensures indistinguishability. More exactly, here, the USS property ensures indistinguishability between a ciphertext obtained from running ScExp and $\mathsf{ReEnc}_{\mathsf{nopk}}$, and a ciphertext obtained from $\mathsf{Enc}_{\mathsf{nopk}}$ and c_{one}. Additionally, by the USS property, the fact that encryption of one output by $\mathcal{F}_{\mathsf{rtprop}}(Z_1)$ to $\mathcal{E}$ is independent from the encryption of one c_{one} given by $\mathcal{A}(Y^*)$ does not allow to distinguish the real and simulated executions.

$\mathcal{S}\mathtt{im}(Z_1)$ proposes the route to $\mathcal{S}\mathtt{im}(Z_2)$ (IV): Let's suppose that, once Z_1 obtains the route towards R^*, $\mathcal{E}$ asks to relay the proposition to an honest node Z_2, by sending input $(\mathsf{proposer}, sid, (Z_2, cid_{Z_1\text{-}Z_2}), PS_{Z_1 \to R^*}, (Y^*, cid_{Z_1\text{-}Y^*}))$ to $\mathcal{F}_{\mathsf{rtprop}}(Z_1)$. Once more, since Z_1 is honest, $\mathcal{E}$ gives the input to $\mathcal{F}_{\mathsf{rtprop}}(Z_1)$, leaving $\mathcal{S}\mathtt{im}$ unaware that it must simulate $\mathcal{S}\mathtt{im}(Z_1)$ proposing a route to $\mathcal{S}\mathtt{im}(Z_2)$. But it receives $(\mathsf{rtprop}, sid, ? \to Z_1 \xrightarrow{cid_{Z_1\text{-}Y^*}} Y^* \xrightarrow{cid_{Y^*\text{-}R^*}} R^*, {dst_{R^*}}^{src_{Z_2}})$ from $\mathcal{F}_{\mathsf{rtprop}}$. $\mathcal{S}\mathtt{im}$ does not need to simulate a transcript between $\mathcal{S}\mathtt{im}(Z_2)$ and $\mathcal{S}\mathtt{im}(Z_1)$, since we are in the $\mathcal{F}_{\mathsf{link}}$-hybrid model (and thus link messages between honest nodes can not be observed by the adversary). However, it needs to simulate a return trip to the end-receiver $\mathcal{A}(R^*)$. For that, $\mathcal{S}\mathtt{im}$ proceeds analogously to the immediately above simulation case, using the value ${dst_{R^*}}^{src_{Z_2}}$, and the c_{one} ciphertext obtained earlier. That is, $\mathcal{S}\mathtt{im}(Z_1)$ sends to $\mathcal{A}(Y^*)$

$$\left\langle \mathtt{rtproprelay} \| cid_{Z_1\text{-}Y^*} \| rcid, \mathsf{Enc}_{\mathsf{nopk}}(c_{\text{one}}, {dst_{R^*}}^{src_{Z_2}}), \mathsf{Enc}_{\mathsf{nopk}}(c_{\text{one}}, pk^{tmp}) \right\rangle$$

with $rcid \leftarrow_{\$} \{0,1\}^*$ and a newly generated key pk^{tmp}. $\mathcal{A}$ carries out the necessary forwarding to $\mathcal{A}(R^*)$, and processes the ciphertexts. Ultimately, at some point,

$\mathcal{A}(R^*)$ will send back data (which would be a pseudonym in a real execution) towards the proposee Z_2, by first going through $\mathcal{A}(Y^*$. Thus, at some point, $\mathcal{S}\mathtt{im}(Z_1)$ will receive a `rtproprelay` message from $\mathcal{A}(Y^*$, with the same *rcid* value, that $\mathcal{S}\mathtt{im}(Z_1)$ simply discards. Lastly, $\mathcal{S}\mathtt{im}$ sends a $\mathsf{continue}$ message to $\mathcal{F}_{\mathsf{rtprop}}$, letting $\mathcal{F}_{\mathsf{rtprop}}(Z_2)$ make its $\mathsf{proposee}$ output to $\mathcal{E}$. This simulation is indistinguishable from a real execution, by the USS property and for the same reason as the previous simulation case.

Note that, when Z_2 is then instructed to further relay further the route proposal to another honest node Z_3 (*i.e.* when $\mathcal{E}$ gives a subsequent $\mathsf{proposer}$ input to $\mathcal{F}_{\mathsf{rtprop}}(Z_2)$), and so on, up to Z_n, the exact same simulations take place. That is, $\mathcal{F}_{\mathsf{rtprop}}$ will perform the same rtprop leak every time (only, with a different ${dst_R}^{src}$ value), and it is still $\mathcal{S}\mathtt{im}(Z_1)$ who will interact with $\mathcal{A}(Y^*)$, because Z_1 is the last honest node before the corrupted end-receiver.

$\mathcal{S}\mathtt{im}(Z_n)$ proposes the route to $\mathcal{A}(X^*)$ (III): Jumping ahead in the scenario, let us suppose that the honest node $\mathcal{S}\mathtt{im}(Z_n)$ proposes the route towards $\mathcal{A}(R^*)$ to a corrupted node $\mathcal{A}(X^*)$. That is, $\mathcal{F}_{\mathsf{rtprop}}(Z_n)$ receives input $(\mathsf{proposer}, sid, (X^*, cid_{X^*\text{-}Z_n}), PS_{X^*\to R^*}, (Z_n, cid_{Z_n\text{-}Z_{n-1}}))$, which $\mathcal{S}\mathtt{im}$ does not see. By construction of the ideal functionality, $\mathcal{S}\mathtt{im}$ gets the leak $(\mathsf{rtprop}, sid, X^* \xleftrightarrow{cid_{X^*\text{-}Z_n}} Z_n \to ?)$ in a first time (when, in $\mathcal{F}_{\mathsf{rtprop}}$, X^* gets the first `rtprop` message). In consequence, $\mathcal{S}\mathtt{im}$ simulates $\mathcal{S}\mathtt{im}(Z_n)$ acting as proposer. For that, it proceeds exactly as though $\mathcal{S}\mathtt{im}(Z_n)$ were *self-proposing* (that is, with $c_{\mathtt{prop}} \coloneqq \mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(Z_n)}, r)$ in particular). At some point $\mathcal{A}(X^*)$ answers with a `rtprop`, from which $\mathcal{S}\mathtt{im}(Z_n)$ extracts and decrypts c_2 to get pk^{tmp}, and then discards the message. At some point (when, in $\mathcal{F}_{\mathsf{rtprop}}$, R^* receives the `rtproprelay` message), $\mathcal{F}_{\mathsf{rtprop}}$ leaks $(\mathsf{rtprop}, sid, X^* \leftrightarrow ? \to Z_1 \to Y^* \to R^*, {dst_{R^*}}^{src_{X^*}})$. Here, $\mathcal{S}\mathtt{im}$ acts as in the immediately above case: $\mathcal{S}\mathtt{im}(Z_1)$ sends $\mathsf{Enc}_{\mathsf{nopk}}(c_{\mathtt{one}}, {dst_{R^*}}^{src_{X^*}})$ and $\mathsf{Enc}_{\mathsf{nopk}}(c_{\mathtt{one}}, pk^{tmp\prime})$ to $\mathcal{A}(R^*)$ in a `rtproprelay` message, and with a fresh key $pk^{tmp\prime}$. When the latter answers, $\mathcal{S}\mathtt{im}(Z_1)$ discards the message. Finally, $\mathcal{F}_{\mathsf{rtprop}}(X)$ outputs $(\mathsf{proposee}, sid, PS_{X^*\to R^*}, {c_{\mathtt{one}}}_{X^*\to R^*}, (Z_n, cid_{X^*\text{-}Z_n}))$ to $\mathcal{S}\mathtt{im}$, which is able to link this with the first rtprop leak thanks to $(Z_n, cid_{X^*\text{-}Z_n})$ notably. Consequently, $\mathcal{S}\mathtt{im}(Z_n)$ answers to $\mathcal{A}(X^*)$ with the message$\langle \mathtt{rtprop} \| cid_{X^*\text{-}Z_n}, \mathsf{Enc}(pk^{tmp}, PS_{X^*\to R^*}), \mathsf{Enc}(pk^{tmp}, 1)\rangle$. This terminates the simulation for this case.

Note how each *half* of the route proposal, the one with X^* and the one with R^*, are handled independently by $\mathcal{S}\mathtt{im}$. Indeed, in the general case, $\mathcal{S}\mathtt{im}$ can (and need) not link together these two halves. Actually, for all that $\mathcal{S}\mathtt{im}$ knows, the first half corresponds to Z_n self-proposing (case (V) in the first scenario); and the second half corresponds to a honest proposer proposing R^* to an honest proposee (case (IV) in the second scenario). Taking a step back, that also means that, when an honest node makes a route proposal, $\mathcal{S}\mathtt{im}$ can only distinguish if it is a proposition towards an honest or corrupted end-receiver *a posteriori* (when it gets the $\mathsf{proposee}$ output with $PS_{X^*\to R^*}$). Also, in future relays of the route proposal, $\mathcal{S}\mathtt{im}(Z_n)$ will fill in for $\mathcal{A}(R^*)$, just as it would do if R^* was honest.

For the first and second halves, taken separately, indistinguishability is already proved since we re-use other simulation cases. Additionally, the fact that the first and second halves are handled independently does not allow to distinguish the simulation from a real execution: the key is that all cryptographic material, and in particular ciphertexts seen by $\mathcal{A}(X^*)$ and $\mathcal{A}(R^*)$, are *expected to be independent* since re-encryptions happen in between. Thus indistinguishability holds by the USS property. Likewise, the key pk^{tmp} generated by $\mathcal{A}(X^*)$ is expected to be independent from the $pk^{tmp\prime}$ as seen by $\mathcal{A}(Y^*)$ and $\mathcal{A}(R^*)$, since each honest relay node multiplies the former by a random temporary public key.

$\mathcal{A}(X^*)$ ***proposes the route again*** *(II)***:** To complete the scenario, let us now assume that $\mathcal{A}(X^*)$ receives input $(\mathsf{proposer}, sid, (X', cid_{X'\text{-}X^*}), PS_{X^*\to R^*}, (X^*, cid_{X^*\text{-}Z_n}))$ from $\mathcal{E}$, instructing $\mathcal{A}(X^*)$ to propose the route it just learned to X'. $\mathcal{S}\mathtt{im}$ sees that input, and knows it relates to a proposition towards the corrupted end-receiver R^*, by the uniqueness of pseudonyms. Consequently, $\mathcal{S}\mathtt{im}$ immediately copies the $\mathsf{proposer}$ input to $\mathcal{F}_{\mathsf{rtprop}}(X^*)$. Two cases arise, depending on the corruption state of X':

– When X' is corrupted, $\mathcal{S}\mathtt{im}(Z_n)$ begins by receiving from $\mathcal{A}(X^*)$ the message

$$\left\langle \mathtt{rtproprelay} \| cid_{X^*\text{-}Z_n} \| rcid, \mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(Z_n)}, r^{src_{X'}}), \mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(Z_n)}, pk^{tmp}) \right\rangle$$

Note that r is the random number that $\mathcal{S}\mathtt{im}(Z_n)$ used in the previous simulation case, to propose the route towards $\mathcal{A}(R^*)$ to $\mathcal{A}(X^*)$. $\mathcal{S}\mathtt{im}$ can trace this message back to $\mathcal{A}(X^*)$ thanks to the cid and $r^{src_{X'}}$ values. $\mathcal{S}\mathtt{im}$ can also know that this `rtproprelay` message relates to the previously seen $\mathsf{proposer}$ input from X^* to X'. Subsequently, $\mathcal{S}\mathtt{im}$ waits to receive $(\mathsf{rtprop}, sid, ? \to Z_1 \to Y^* \to R^*, {dst_{R^*}}^{src_{X'}})$, which is leaked by $\mathcal{F}_{\mathsf{rtprop}}$ when $\mathcal{F}_{\mathsf{rtprop}}(R^*)$ receives the `rtproprelay` message in the ideal functionality. Once received, $\mathcal{S}\mathtt{im}(Z_1)$ interacts with $\mathcal{A}(Y^*)$ in the exact same way as in the *second half* in the preceding simulation case (again, $\mathcal{S}\mathtt{im}$ does not necessarily know to which route proposal this rtprop leaks relates, but it does not matter). When $\mathcal{A}(Y^*)$ sends back a `rtproprelay` message, $\mathcal{S}\mathtt{im}(Z_1)$ discards it. At some point, $\mathcal{F}_{\mathsf{rtprop}}$ will leak $(\mathsf{rtprop}, sid, X' \xleftarrow{cid_{X'\text{-}X^*}} X^* \xrightarrow{cid_{X^*\text{-}Z_n}} Z_n \to ? \to R^*)$, namely when $\mathcal{F}_{\mathsf{rtprop}}(X^*)$ sends the last `rtprop` message to $\mathcal{F}_{\mathsf{rtprop}}(X')$ in the ideal functionality. Simultaneously, $\mathcal{F}_{\mathsf{rtprop}}(X')$ will output $(\mathsf{proposee}, sid, PS_{X\to R^*}, c_{\mathtt{one}R^*}, (X^*, cid_{X'\text{-}X^*}))$. $\mathcal{S}\mathtt{im}$ can thus identify $PS_{X'\to R^*}$ as the pseudonym it must send back toward X', in particular thanks to $(X^*, cid_{X'\text{-}X^*})$. Proceeding as usual, $\mathcal{S}\mathtt{im}(Z_n)$ sends:

$$\left\langle \mathtt{rtproprelay} \| cid_{X^*\text{-}Z_n} \| rcid, \mathsf{Enc}(pk^{tmp}, PS_{X'\to R^*}), \mathsf{Enc}(pk^{tmp}, 1) \right\rangle$$

to $\mathcal{A}(X^*)$, with $rcid$ and pk^{tmp} as received by $\mathcal{S}\mathtt{im}(Z_n)$ earlier.

– When X' is honest, $\mathcal{S}\mathtt{im}(X')$ begins by receiving a `rtprop` message from $\mathcal{A}(X^*)$. Similarly to previous cases, $\mathcal{S}\mathtt{im}(X')$ answers using a random $r' \leftarrow_{\$} \mathbb{G}$

instead of $src_{X'}$. At some point, $\mathcal{S}\mathtt{im}(Z_n)$ receives a rtproprelay message as in the case when X' is corrupted. Then, the *second half* of the route, with Z_1, Y^* and R^*, is simulated as usual. Later, when in the ideal functionality, node X^* sends the last `rtprop` message, $\mathcal{F}_{\mathsf{rtprop}}$ leaks $(\mathsf{rtprop}, sid, X' \xleftarrow{cid_{X'\text{-}X^*}} X^* \xrightarrow{cid_{X^*\text{-}Z_n}} Z_n \rightarrow ? \rightarrow R^*)$. $\mathcal{S}\mathtt{im}$ however does not get to see $\mathcal{F}_{\mathsf{rtprop}}(X')$'s proposee output, which includes $PS_{X' \rightarrow R^*}$. Still, it can safely make $\mathcal{S}\mathtt{im}(Z_n)$ send back a `rtproprelay` message with $c_1 \coloneqq \mathsf{Enc}(pk^{tmp}, r^{r'})$ to $\mathcal{A}(X^*)$. The latter will forward it to $\mathcal{S}\mathtt{im}(X')$, who can discard the message. Lastly, $\mathcal{S}\mathtt{im}$ sends a continue message to $\mathcal{F}_{\mathsf{rtprop}}$, letting $\mathcal{F}_{\mathsf{rtprop}}(X')$ make its proposee output to $\mathcal{E}$.

Since this simulation case is only a re-use of previously presented simulation methodologies, indistinguishability for the first and second halves separately is already proven. Now, by the same argument as the previous simulation case, and mainly thanks to re-encryptions, the fact that the two halves are totally independently simulated is indistinguishable from a real execution, by the USS property.

This concludes the second scenario. With these two scenarios, all cases of interest are covered. There may be some slight differences, *e.g.* a variant of the second scenario could include corrupted nodes between, say, $\mathcal{S}\mathtt{im}(Z_3)$ and $\mathcal{S}\mathtt{im}(Z_2)$. However, such intermediary corrupted sub-paths can always be safely simulated as presented below.

Formal simulation of intermediary corrupted sub-paths: At the beginning of this proof, we informally described how intermediary sub-paths were to be simulated. Here, now that we have formally defined the simulations of *end* sub-paths, we provide a formal description. When $\mathcal{S}\mathtt{im}$ receives

$$(\mathsf{subpath}, sid, Z_{i'} \xrightarrow[rcid_{i'+1}]{cid_{i'+1}} \dots \xrightarrow[rcid_{j'}]{cid_{j'}} Z_{j'})$$

from $\mathcal{F}_{\mathsf{rtprop}}$, it necessarily corresponds to a sequence of corrupted nodes forwarding a `rtproprelay` message, in which $Z_{i'}$ and $Z_{j'}$ are honest nodes. This sub-path can correspond to the *forward* or *backward* direction of the return trip (*i.e.* either the outward journey or or the return journey of the `rtproprelay` message). Since the simulation of the backward direction depends on how the forward direction was simulated, we first describe the simulation of a sub-path corresponding to the forward relay.

By the construction of the simulation, note that $\mathcal{S}\mathtt{im}(Z'_i)$ is sure to know a correct encryption of one $c_{\mathtt{one}} = \mathsf{Enc}(pk_{Z_{i'+1},\dots,Z_{j'-1},\mathcal{S}\mathtt{im}(Z_{j'})}, 1)$. $\mathcal{S}\mathtt{im}(Z_{i'})$ encrypts $r \leftarrow\!\!\$\ \mathbb{G}$ and a freshly generated key pk^{tmp} in c_1 and c_2 respectively, using $\mathsf{Enc}_{\mathsf{nopk}}$ and $c_{\mathtt{one}}$. Then, $\mathcal{S}\mathtt{im}(Z_{i'})$ sends $\langle \mathtt{rtproprelay} \| cid_{i'+1} \| rcid, c_1, c_2 \rangle$ to $\mathcal{A}(Z_{i'+1})$, with $rcid \leftarrow\!\!\$\ \{0,1\}^*$. By the construction of the simulation, the message necessarily

traverses $\mathcal{A}(Z_{i'+1})$, $\mathcal{A}(Z_{i'+2})$, *etc.*, and necessarily arrives at $\mathcal{S}\mathtt{im}(Z_{j'})$ as

$$\left\langle \mathtt{rtproprelay} \| cid_{j'} \| rcid', \mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(Z_{j'})}, r), \mathsf{Enc}(pk_{\mathcal{S}\mathtt{im}(Z_{j'})}, pk^{tmp\prime}) \right\rangle$$

The latter can decrypt the ciphertexts, recognise the random value r, and discard the message. However, $\mathcal{S}\mathtt{im}(Z_{j'})$ stores $(rcid_{j'}, rcid', pk^{tmp\prime})$, so as to simulate the return path.

Now, when $\mathcal{S}\mathtt{im}$ receives $(\mathsf{subpath}, sid, Z_{j'} \xrightarrow[rcid_{j'}]{cid_{j'}} \dots \xrightarrow[rcid_{i'+1}]{cid_{i'+1}} Z_{i'})$, with cid used in reverse fashion, and such that the corresponding *forward* path was simulated previously, $\mathcal{S}\mathtt{im}$ proceeds as follows. It first retrieves the stored tuple associated to $rcid_{j'}$ to get $rcid'$ and $pk^{tmp\prime}$. That is, $rcid_{j'}$ is what *binds* the forward and return sub-paths leaked by $\mathcal{F}_{\mathsf{rtprop}}$ (as it would in a actual network nodes in a real execution). Then, $\mathcal{S}\mathtt{im}(Z_{j'})$ sends

$$\left\langle \mathtt{rtproprelay} \| cid_{j'} \| rcid', \mathsf{Enc}(pk^{tmp\prime}, r), \mathsf{Enc}(pk^{tmp\prime}, 1) \right\rangle$$

to $\mathcal{A}(Z_{j'-1})$. The message necessarily returns to $\mathcal{S}\mathtt{im}(Z_{i'})$, which can not decrypt the ciphertexts, but recognises the $rcid$ value and discards the message.

The simulation of intermediary corrupted sub-paths is indistinguishable from a real execution, by the IND-CPA, IK-CPA and USS properties. In particular, the fact that ciphertexts seen by the corrupted relay nodes are independent from any other ciphertext in the network (since they are *crafted* by the simulator, and encrypt a random plaintext) is indistinguishable from a real execution by the USS property. Finally, as mentioned previously, all sub-paths are simulated in the order they are received from $\mathcal{F}_{\mathsf{rtprop}}$, and more generally, throughout the entire simulation, care was taken to enforce the right ordering of events.

This terminates the description of the simulator. To show indistinguishability, a couple of arguments remain to be specified, for the simulation as a whole. First, $\mathcal{S}\mathtt{im}$ relies on the randomness of the $rcid$ values or of values such as $r^{r'}$. The simulation fails if *e.g.* the same $rcid$ value is used twice. However, the size of the sets from which these values are taken from is exponential in the security parameter, and hence this happens with negligible probability. Lastly, indistinguishability holds because the communications between $\mathcal{A}$ and $\mathcal{E}$ are relayed by $\mathcal{S}\mathtt{im}$ without alteration, and because Π_{rtprop} and $\mathcal{F}_{\mathsf{rtprop}}$ exhibit the same input/output behavior. This concludes the proof that the simulated and real executions are indistinguishable. □

B.4. Analysis of $\mathcal{F}_{\mathsf{rtprop}}$

This section provides supplementary material with respect to the analysis of the route proposal protocol in Section 5.5.3 (p. 115). Namely, we present the explicit adversarial views from Definition 25, and the proof that the proposed route proposal mechanism does fulfill the said definition.

B.4.1. Explicit Adversarial Views

We begin by explicitly describing the view of the adversary for each of the four properties presented in Definition 25 (p. 119). Each view is given for R_b, $b \in \{0,1\}$, and the elements that vary depending on the value of b are in bold font. Note that in each view, there is redundant information (*e.g.* $PS_{X \to R_b}$ appears twice in the view for route proposal homogeneity), but this does not impact the proof.

Route Proposal Homogeneity The most constraining case is when X is corrupted (otherwise, the view is empty, since by definition, Y and R are honest). Denote $R_0 :- Y$ and $R_1 := R$. The context and view are as follow:

$$\mathsf{Context}^{\{X\}}_{Y,R}(i) = \begin{cases} \{(Z, src_Z, dst_Z, (pk_Z, sk_Z)) \mid Z \in \Omega_c\} \\ \cup\ \{(Z, R', PS_{Z \to R'}, c_{\mathsf{one}\,Z \to R'}, cid_{Z \to R'}) \\ \qquad\qquad \mid Z \in \Omega_c, R' \in \Omega \setminus \{Y, R\}\} \\ \cup\ \{(Z, (R_a, PS_{Z \to R_a}, c_{\mathsf{one}\,Z \to R_a}, cid_{Z \to R_a})) \\ \qquad\qquad \mid a \in \{0,1\}, Z \in \Omega_c \setminus \{X\}\} \end{cases}$$

$$\mathsf{View}^{\{X\}}_{\mathtt{RP}(X \leftrightarrow Y \to R_b)}(i) = \begin{cases} (\mathsf{rtprop}, sid, X \xleftrightarrow{\mathbf{cid}} Y \to ?), \\ (\mathsf{proposee}, sid, \mathbf{PS_{X \to R_b}}, \mathbf{c_{one\,X \to R_b}}, (Y, \mathbf{cid})) \\ (X, \mathbf{PS_{X \to R_b}}, \mathbf{c_{one\,X \to R_b}}) \end{cases}$$

Route Proposal Indistinguishability The most *constraining* case, here, is when X and Y are both corrupt (otherwise, the adversary only gets less information). In this case, the context and view are as follows, for $\mathcal{N} := \{X, Y\} \cup \mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K$:

$$\mathsf{Context}^{\mathcal{N}}_{R_0,R_1}(i) = \begin{cases} \{(Z, src_Z, dst_Z, (pk_Z, sk_Z)) \mid Z \in \Omega_c\} \\ \cup\ \{(Z, R', PS_{Z \to R'}, c_{\mathsf{one}\,Z \to R'}, cid_{Z \to R'}) \\ \qquad\qquad \mid Z \in \Omega_c, R' \in \Omega \setminus \{R_0, R_1\}\} \\ \cup\ \{(Z, (R_a, PS_{Z \to R_a}, c_{\mathsf{one}\,Z \to R_a}, cid_{Z \to R_a})) \\ \qquad\qquad \mid a \in \{0,1\}, Z \in (\Omega_c \setminus \mathcal{N})\} \end{cases}$$

$$\mathsf{View}^{\mathcal{N}}_{\mathtt{RP}(X \leftrightarrow Y \to R_b)}(i) = \begin{cases} (\mathsf{proposer}, sid, (X, \mathbf{cid}), \mathbf{PS_{Y \to R_b}}, (Z_{1,1}, \mathbf{cid_{1,1}})) \\ (\mathsf{rtprop}, sid, X \xleftrightarrow{\mathbf{cid}} Y \xrightarrow{\mathbf{cid_{1,1}}} (Z_{1,1} \xrightarrow{\mathbf{cid_{1,2}}} \cdots \xrightarrow{\mathbf{cid_{1,n_1}}} Z_{1,n_1}) \to ?), \\ (\mathsf{subpath}, sid, sp_2 = (Z_{2,1} \xrightarrow{\mathbf{cid_{2,2}}} \cdots \xrightarrow{\mathbf{cid_{2,n_2}}} Z_{2,n_2})), \\ \ldots, \\ (\mathsf{subpath}, sid, sp_K), \\ (\mathsf{proposee}, sid, \mathbf{PS_{X \to R_b}}, \mathbf{c_{one\,X \to R_b}}, (Y, \mathbf{cid}))) \\ \forall Z \in \mathcal{N} \cap \Omega_c, (Z, \mathbf{PS_{Z \to R_b}}, \mathbf{c_{one\,Z \to R_b}}) \end{cases}$$

Untraceable Propagation The most constraining case is when X, Y, X', Y' are all corrupted (otherwise, the adversary only gets less information). In this case, the context

and view are as follows, for $\mathcal{N} \coloneqq \{X, Y\} \cup \mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K$ and $\mathcal{N}' \coloneqq \{X', Y'\} \cup \mathcal{Z}'_1 \cup \cdots \cup \mathcal{Z}'_K$:

$$\mathsf{Context}^{\mathcal{N}\cup\mathcal{N}'}_{R_0,R_1}(i) = \begin{cases} \{(Z, src_Z, dst_Z, (pk_Z, sk_Z)) \mid Z \in \Omega_c\} \\ \cup\ \{(Z, R', PS_{Z\to R'}, c_{\mathtt{one}Z\to R'}, cid_{Z\to R'}) \\ \qquad\qquad \mid Z \in \Omega_c, R' \in \Omega \setminus \{R_0, R_1\}\} \\ \cup\ \{(Z, (R_a, PS_{Z\to R_a}, c_{\mathtt{one}Z\to R_a}, cid_{Z\to R_a})) \\ \qquad\qquad \mid a \in \{0,1\}, Z \in \Omega_c \setminus (\mathcal{N} \cup \mathcal{N}')\} \end{cases}$$

$$\mathsf{View}^{\mathcal{N}}_{\mathtt{RP}(X\leftrightarrow Y\to R_0)}(i) = \begin{cases} (\mathsf{proposer}, sid, (X, cid), PS_{Y\to R_0}, (Z_{1,1}, cid_{1,1})) \\ (\mathsf{rtprop}, sid, X \overset{cid}{\longleftrightarrow} Y \xrightarrow{cid_{1,1}} (Z_{1,1} \overset{cid_{1,2}}{\to} \ldots \overset{cid_{1,n_1}}{\to} Z_{1,n_1}) \to ?), \\ (\mathsf{subpath}, sid, sp_2 = (Z_{2,1} \overset{cid_{2,2}}{\to} \ldots \overset{cid_{2,n_2}}{\to} Z_{2,n_2})), \\ \ldots, \\ (\mathsf{subpath}, sid, sp_K), \\ (\mathsf{proposee}, sid, PS_{X\to R_0}, c_{\mathtt{one}X\to R_0}, (Y, cid))) \\ \forall Z \in \mathcal{N} \cap \Omega_c, (Z, PS_{Z\to R_0}, c_{\mathtt{one}Z\to R_0}) \end{cases}$$

$$\mathsf{View}^{\mathcal{N}'}_{\mathtt{RP}(X'\leftrightarrow Y'\to R_b)}(i) = \begin{cases} (\mathsf{proposer}, sid, (X', \mathbf{cid}'), \mathbf{PS}_{\mathbf{Y}'\to\mathbf{R_b}}, (Z'_{1,1}, \mathbf{cid}'_{\mathbf{1,1}})) \\ (\mathsf{rtprop}, sid, X' \overset{\mathbf{cid}'}{\longleftrightarrow} Y' \xrightarrow{\mathbf{cid}'_{\mathbf{1,1}}} (Z'_{1,1} \overset{\mathbf{cid}'_{\mathbf{1,2}}}{\to} \ldots \overset{\mathbf{cid}'_{\mathbf{1,n'_1}}}{\to} Z'_{1,n'_1}) \to ?), \\ (\mathsf{subpath}, sid, sp'_2 = (Z'_{2,1} \overset{\mathbf{cid}'_{\mathbf{2,2}}}{\to} \ldots \overset{\mathbf{cid}'_{\mathbf{2,n'_2}}}{\to} Z'_{2,n'_2})), \\ \ldots, \\ (\mathsf{subpath}, sid, sp'_{K'}), \\ (\mathsf{proposee}, sid, \mathbf{PS}_{\mathbf{X}'\to\mathbf{R_b}}, \mathbf{c}_{\mathtt{one}\mathbf{X}'\to\mathbf{R_b}}, (Y', \mathbf{cid}'))) \\ \forall Z \in \mathcal{N}' \cap \Omega_c, (Z, \mathbf{PS}_{\mathbf{Z}\to\mathbf{R_b}}, \mathbf{c}_{\mathtt{one}\mathbf{Z}\to\mathbf{R_b}}) \end{cases}$$

Untraceable Return Trip The main case of interest is when X_0, Y_0, X_1, and Y_1 are all corrupted, and $R_0 \neq R_1 \in \Omega_{\mathcal{S}\mathtt{im}}$. Indeed, note that when $R_0 = R_1$, the challenge view is actually the same whether $b = 0$ or $b = 1$, up to the *cid* values (which do not allow to distinguish). In the case when $R_0 \neq R_1 \in \Omega_{\mathcal{S}\mathtt{im}}$, the context and view are as follow, for $\mathcal{N} \coloneqq \{X_0, Y_0\} \cup \mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K$ and $\mathcal{N}' \coloneqq \{X_1, Y_1\} \cup \mathcal{Z}'_1 \cup \cdots \cup \mathcal{Z}'_{K'}$:

$$\mathsf{Context}^{\mathcal{N}\cup\mathcal{N}'}_{R_0,R_1}(i) = \begin{cases} \{(Z, src_Z, dst_Z, (pk_Z, sk_Z)) \mid Z \in \Omega_c\} \\ \cup\ \{(Z, R', PS_{Z\to R'}, c_{\mathtt{one}Z\to R'}, cid_{Z\to R'}) \\ \qquad\qquad \mid Z \in \Omega_c, R' \in \Omega \setminus \{R_0, R_1\}\} \\ \cup\ \{(Z, (R_a, PS_{Z\to R_a}, c_{\mathtt{one}Z\to R_a}, cid_{Z\to R_a})) \\ \qquad\qquad \mid a \in \{0,1\}, Z \in \Omega_c \setminus (\mathcal{N} \cup \mathcal{N}')\} \end{cases}$$

$$\mathsf{View}^{\mathcal{N}\setminus(\mathcal{Z}_{k_1}\cup\cdots\cup\mathcal{Z}_{k_2})}_{\mathtt{RP}(X_0\leftrightarrow Y_0\rightarrow R_0)}(i) = \begin{cases} (\mathsf{proposer}, sid, (X_0, cid), PS_{Y_0\rightarrow R_0}, (Z_{1,1}, cid_{1,1})) \\ (\mathsf{rtprop}, sid, X_0 \overset{cid}{\longleftrightarrow} Y_0 \xrightarrow{cid_{1,1}} (Z_{1,1} \overset{cid_{1,2}}{\rightarrow} \dots \overset{cid_{1,n_1}}{\rightarrow} Z_{1,n_1}) \rightarrow ?), \\ (\mathsf{subpath}, sid, sp_2 = (Z_{2,1} \overset{cid_{2,2}}{\rightarrow} \dots \overset{cid_{2,n_2}}{\rightarrow} Z_{2,n_2})), \\ \dots, \\ (\mathsf{subpath}, sid, sp_{k_1-1}), (\mathsf{subpath}, sid, sp_{k_2+1}), \\ \dots, \\ (\mathsf{subpath}, sid, sp_K), \\ (\mathsf{proposee}, sid, PS_{X_0\rightarrow R_0}, c_{\mathsf{one}X_0\rightarrow R_0}, (Y_0, cid))) \\ \forall Z \in (\mathcal{N} \setminus (\mathcal{Z}_{k_1} \cup \cdots \cup \mathcal{Z}_{k_2})) \cap \Omega_c, (Z, PS_{Z\rightarrow R_0}, c_{\mathsf{one}Z\rightarrow R_0}) \end{cases}$$

$$\mathsf{View}^{\mathcal{Z}'_{k'_1}\cup\cdots\cup\mathcal{Z}'_{k'_2}}_{\mathtt{RP}(X_b\leftrightarrow Y_b\rightarrow R_b)}(i) = \begin{cases} (\mathsf{subpath}, sid, sp_{k_1} = (Z_{k_1,1} \overset{\mathbf{cid_{k_1,2}}}{\rightarrow} \dots \overset{\mathbf{cid_{k_1,n_2}}}{\rightarrow} Z_{k_1,n_2})), \\ \dots, \\ (\mathsf{subpath}, sid, sp'_{k_2}), \\ \forall Z \in (\mathcal{Z}'_{k'_1} \cup \cdots \cup \mathcal{Z}'_{k'_2}) \cap \Omega_c, (Z, \mathbf{PS_{Z\rightarrow R_b}}, \mathbf{c_{oneZ\rightarrow R_b}}) \end{cases}$$

B.4.2. Proof of Theorem 3 (Route Proposal Security)

Given these explicit adversarial view, we now prove Theorem 3 (p. 119), which states the security of route proposals.

Proof of Theorem 3. For each property, we show that, the adversary's advantage in distinguishing between the two view distributions, is, for $k, k' \in poly(\lambda)$, at most:

$$k \cdot \mathsf{Adv}^{\mathsf{ik\text{-}cpa}}_{\mathcal{B}}(\lambda) + k' \cdot \mathsf{Adv}^{\mathsf{ps-ind}}_{\mathcal{B}}(\lambda)$$

Namely, $(k, k') = (1, 1)$ for route proposal homogeneity, $(k, k') = (l_{max}, l_{max})$ for route proposal indistinguishability, propagation untraceability, and return trip untraceability.

We provide the full proof for route proposal indistinguishability. The proofs for the other properties follow the same outline. In what follows, denote by:

$$\mathcal{N} \coloneqq \{X, Y, Z_{1,1}, \dots, Z_{k,n_k}\} \cap \Omega_c = (N_1, \dots, N_i, N_{i+1}, \dots, N_n)$$

In light of the view for route proposal indistinguishability property, we show that, given $\mathsf{Context}^{\mathcal{N}}_{R_0,R_1}(i)$, the advantage of $\mathcal{A}$ in distinguishing $\mathsf{View}^{\mathcal{N}}_{\mathtt{RP}(X\leftrightarrow Y\rightarrow R_0)}(i)$ from $\mathsf{View}^{\mathcal{N}}_{\mathtt{RP}(X\leftrightarrow Y\rightarrow R_1)}(i)$, for a given $i \in I^{RPI}_W$, is bounded above by

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) \leq |\mathcal{N}|\, \mathsf{Adv}^{\mathsf{ik\text{-}cpa}}_{\mathcal{B}}(\lambda) + |\mathcal{N}|\, \mathsf{Adv}^{\mathsf{ps-ind}}_{\mathcal{B}}(\lambda)$$

The theorem directly follows, since, by definition $\mathcal{N} \leq l_{max}$. To prove the above claim, we use the two following hybrid arguments, linked together.

Hybrid $\mathcal{H}^{(i)}_{\texttt{cone}}$ for $i \in [0, |\mathcal{N}|]$: $\mathcal{H}^{(0)}_{\texttt{cone}}$ consists in distinguishing the original definition of the views, $\mathcal{H}^{(1)}_{\texttt{cone}}$ consists in distinguishing the views where $c_{\texttt{cone}X \to R_b}$ was replaced by an encryption of one under a random public key, $\mathcal{H}^{(2)}_{\texttt{cone}}$ consists in distinguishing the views where $c_{\texttt{cone}Y \to R_b}$ was also replaced by an encryption of one under a random (different) public key, etc. Finally, $\mathcal{H}^{(|\mathcal{N}|)}_{\texttt{cone}}$ consists in distinguishing the views where all encryptions of one were replaced by encryptions of one under a different, random public key each.

Hybrid $\mathcal{H}^{(i)}_{PS}$ for $i \in [0, |\mathcal{N}|]$: $\mathcal{H}^{(0)}_{PS} = \mathcal{H}^{(i)}_{\texttt{cone}}$, $\mathcal{H}^{(1)}_{PS}$ consists in distinguishing the view where $PS_{X \to R_b}$ was replaced a pseudonym $PS_{X \to rand} = \mathsf{h}(dst^{src_X})$ with a random dst, $\mathcal{H}^{(2)}_{PS}$ consists in distinguishing the view where $PS_{Y \to R_b}$ was also replaced a (different) pseudonym $PS_{Y \to rand} = \mathsf{h}(dst'^{src}X)$ with a random dst' (independent from dst), etc. Finally, $\mathcal{H}^{(|\mathcal{N}|)}_{PS}$ consists in distinguishing the views where pseudonyms were replaced by pseudonyms computed with a different, random dst value each.

The advantage of $\mathcal{A}$ in the last hybrid $\mathcal{H}^{(|\mathcal{N}|)}_{PS}$ is zero. Indeed, (i) the *cid* values, pseudonyms, and encryptions of one are essentially random values, not giving any information on R_b, and (ii) note additionally that by definition, $\mathcal{Z}_K$ can not terminate with $R_{b'}$ (unless $R_0 = R_1$). We then show that the advantage of the adversary in distinguishing neighboring hybrids is either $\mathsf{Adv}^{\mathsf{ik\text{-}cpa}}_{\mathcal{B}}(\lambda)$, or $\mathsf{Adv}^{\mathsf{ps-ind}}_{\mathcal{B}}(\lambda)$, which in turn yield the desired result.

To prove indistinguishability between each adjacent hybrid games $\mathcal{H}^{(i)}_{\texttt{cone}}$ and $\mathcal{H}^{(i+1)}_{\texttt{cone}}$, let us assume that there exists an adversary $\mathcal{A}$ capable of distinguishing between them with advantage $\mathsf{Adv}^{\mathcal{H}^{(i/i+1)}_{\texttt{cone}}}_{\mathcal{A}}(\lambda)$. Then, we construct an adversary $\mathcal{B}$ that, using $\mathcal{A}$ breaks the OK-CPA property of the Elgamal scheme with the same advantage. $\mathcal{B}$ takes in input two public keys pk_0, pk_1. To generate all the elements of the view that $\mathcal{A}$ expects, $\mathcal{B}$ begins by sampling $src_Z, dst_Z \leftarrow_{\$} \mathbb{Z}^*_q \times \mathbb{G}$ for all $Z \in \Omega$, and $(pk_Z, sk_Z) \leftarrow \mathsf{KeyGen}(1^\lambda)$ for all $Z \in \Omega \setminus \{R_0, R_1\}$. For the end-receivers R_0 and R_1, $\mathcal{B}$ samples $b' \leftarrow_{\$} \{0, 1\}$, and assigns $pk_{R_{b'}} \coloneqq pk_0$, and $(pk_{R_{1-b'}}, sk_{R_{1-b'}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. Then, $\mathcal{B}$ selects honest nodes to *fill the gap* between each intermediary corrupted sub-paths $\mathcal{Z}_k$, and between the last corrupted sub-path $\mathcal{Z}_K$ and the potential end-receivers R_0 and R_1. $\mathcal{B}$ thus obtains full routes, from Y to R_0 and from Y to R_1, respectively denoted rt_0 and rt_1.

From these routes, $\mathcal{B}$ can generate all the necessary material. Hereby, for simplicity, $\mathcal{N}$ is considered as the set $\left\{N_1, \ldots, N_{|\mathcal{N}|}\right\}$, in particular containing N_{i+1}, which is the pivotal point of hybrid $\mathcal{H}^{(i+1)}_{\texttt{cone}}$. $\mathcal{B}$ consequently generates:

- $PS_{Z \to R_0} = \mathsf{h}({dst_{R_0}}^{src_Z})$, and $PS_{Z \to R_1} = \mathsf{h}({dst_{R_1}}^{src_Z})$ for all $Z \in \Omega$
- For all nodes $N \in \{N_1, \ldots, N_i\}$, $c_{\texttt{cone}N \to R_{b'}} \leftarrow \mathsf{Enc}(pk, 1)$ for a freshly generated public key pk (different for each N)
- For all nodes $N \in \left\{N_{i+2}, \ldots, N_{|\mathcal{N}|}\right\}$, $c_{\texttt{cone}N \to R_{b'}} \leftarrow \mathsf{Enc}(pk_{rt_{b'}[N\text{-}R_{b'}]}, 1)$ and for $pk_{rt_{b'}[N\text{-}R_{b'}]}$ the product of public key of all nodes between N (excluded) and $R_{b'}$

(included) on route $rt_{b'}$

– For all remaining corrupted nodes $Z \in \Omega_c \setminus \mathcal{N}$, set $c_{\mathtt{one}Z \to R_0} \leftarrow \mathsf{Enc}(pk_{rt'[Z\text{-}R_0]}, 1)$ and $c_{\mathtt{one}Z \to R_1} \leftarrow \mathsf{Enc}(pk_{rt''[Z-R_1]}, 1)$ for some route rt' and rt'' from Z to R_0 and R_1 (respectively), randomly selected by $\mathcal{B}$.

For the case of N_{i+1}, $\mathcal{B}$ outputs $m = 1$ as its challenge, and receives c_b. It sets $c_{\mathtt{one}N_{i+1} \to R_{b'}} = \mathsf{KeyMult}(sk, c_b)$, where sk is the sum of secret keys of nodes between N_{i+1} and $R_{b'}$ (both excluded) on route rt. Finally, $\mathcal{B}$ gives to $\mathcal{A}$ all the generated elements, according to what $\mathcal{A}$ expects as context and view, and in particular: $PS_{N \to R_{b'}}$, $PS_{N \to R_{b'}}$, and $c_{\mathtt{one}N \to R_{b'}}$ for each $N \in \mathcal{N}$. $\mathcal{A}$ makes its guess: if it outputs "$\mathcal{H}^{(i)}$", $\mathcal{B}$ outputs $b^* = 0$, and $b^* = 1$ if $\mathcal{A}$ outputs "$\mathcal{H}^{(i+1)}$". $\mathcal{B}$ breaks the key privacy property with the same advantage that $\mathcal{A}$ has, since: (i) all the elements of the view apart from the encryption of one $c_{\mathtt{one}N_{i+1} \to R_{b'}}$ of node N_{i+1} are well generated/simulated, and (ii) when $b = 0$, $c_{\mathtt{one}N_{i+1} \to R_{b'}}$ is $\mathsf{Enc}(pk_{Z-R_{b'}}, 1)$, because $pk_{R_{b'}} = pk_0$, and when $b = 1$, it is an encryption of one under $(pk_{Z-R_{b'}}/pk_{R_{b'}}) \cdot pk_1$ which is random to $\mathcal{A}$, since pk_1 does not appear anywhere else in the elements given to her.

To prove the indistinguishability of each adjacent hybrid games $\mathcal{H}^{(i)}_{PS}$ and $\mathcal{H}^{(i+1)}_{PS}$, the same methodology applied. There are a few differences, however. First, all encryptions of one for nodes in $\mathcal{N}$ (those receiving the challenge), are made under different, random public keys. The challenge of $\mathcal{B}$ consisting of (src, PS_1, PS_2) is bound to $\mathcal{A}$'s challenge in the following way. First, the value $src_{N_{i+1}}$ is set to src. Then, pseudonyms towards R_1 are computed directly by $\mathcal{B}$, using a randomly generated dst_{R_1} value, while for pseudonyms towards R_0 are computed using her oracle $f(\cdot, dst_0)$. The exceptions being that: the pseudonyms towards $R_{b'}$ for each node $N_1, \ldots, N_i$ are computed directly and with a different, random dst value each; and $PS_{N_{i+1} \to R_{b'}}$ is set to PS_1. Finally, if $\mathcal{A}$ outputs "$\mathcal{H}^{(i)}$", $\mathcal{B}$ outputs $b^* = 0$, and $b^* = 1$ otherwise. $\mathcal{B}$ breaks the indistinguishability of pseudonyms with the same advantage as $\mathcal{A}$, since: (i) all the elements of the view apart from N_{i+1}'s pseudonyms towards the end-receiver are well generated/simulated, and (ii) when $b = 0$, $PS = \mathsf{h}(dst_0^{src}) = PS_{N_{i+1} \to R_{b'}}$, and when $b = 1$, $PS = \mathsf{h}(dst_1^{src})$ which is a pseudonym with a dst value that never appears elsewhere in the view, and thus a random one.

Putting hybrids together, since $\mathcal{H}^{(0)}_{PS} = \mathcal{H}^{(i)}_{\mathtt{cone}}$, we have that:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) \leq |\mathcal{N}|\, \mathsf{Adv}^{\mathsf{ik\text{-}cpa}}_{\mathcal{B}}(\lambda) + |\mathcal{N}|\, \mathsf{Adv}^{\mathsf{ps-ind}}_{\mathcal{B}}(\lambda)$$

Since the same reasoning can be applied to any input $i \in I^{RPI}_W$, this concludes the proof for the case of route proposal indistinguishability.

For the propagation and return trip untraceability, the same methodology with two hybrid sequences can be applied, yielding the same results. For the route proposal homogeneity property, $\mathcal{A}$'s advantage is smaller, since only *one* pseudonym and encryption of one differ following the value of b (those of X's towards R_b). Said otherwise, the hybrid sequences are actually of size one. This concludes the proof for Theorem 3. □

B.5. Proof of Theorem 4 (Π UC-Realizes $\mathcal{F}$)

Below is the proof showing that the full protocol Π (defined p. 122) UC-realises $\mathcal{F}$ (defined p. 125). All the comments made in Section B.2 about the UC framework, and the distinction between $\mathcal{E}$ and $\mathcal{A}$ also apply here. In particular, we denote corrupted nodes played by adv as $\mathcal{A}(X)$, the honest nodes simulated by $\mathcal{S}\mathtt{im}$ as $\mathcal{S}\mathtt{im}(X)$, and the dummy parties as $\mathcal{F}_{\mathsf{rtprop}}(X)$.

Proof of Theorem 4. Let $\mathcal{A}$ be a semi-honest, static adversary in a real execution that interacts with parties running the protocol Π in the $(\mathcal{F}_{\text{reg}}, \mathcal{F}_{\mathsf{link}}, \mathcal{F}_{\mathsf{offline}}, \mathcal{F}_{\mathsf{rtprop}})$-hybrid model. We construct a simulator $\mathcal{S}\mathtt{im}$ in the ideal execution, interacting with $\mathcal{F}$ in the $\mathcal{F}_{\text{reg}}$-hybrid model, such that no environment $\mathcal{E}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and the protocol, or with $\mathcal{S}\mathtt{im}$ and the ideal functionality. Figure 5.11 in Section 5.6 (p. 126) shows the setup and relations between ITIs to carry out the proof. As usual, the simulator $\mathcal{S}\mathtt{im}$ starts by invoking a copy of $\mathcal{A}$. $\mathcal{S}\mathtt{im}$ also internally runs a copy of $\mathcal{F}_{\mathsf{link}}$ and $\mathcal{F}_{\mathsf{offline}}$. However, $\mathcal{F}_{\text{reg}}$ stays outside of $\mathcal{S}\mathtt{im}$, and $\mathcal{F}_{\mathsf{rtprop}}$ is run internally by $\mathcal{F}$.

We describe how $\mathcal{S}\mathtt{im}$ simulates a real execution, by acting on behalf of the honest nodes X that interact with $\mathcal{A}$, and show that this simulation is indistinguishable from a real execution. After a few preliminaries, the description of the simulator (and the proof itself) is presented as follows. First, the simulation of topology dissemination is presented. Then the case of payload messages sent by corrupted end-senders is considered and divided in three main steps corresponding to: the offline interaction between end-sender and end-receiver, the *first leg* between end-sender and indirection node, and the *second leg* between indirection node and end-receiver. And finally, the simulation of messages sent by honest end-senders is presented, analogously separated in three main steps.

Simulating the communication between $\mathcal{E}$ and $\mathcal{A}$: $\mathcal{S}\mathtt{im}$ internally runs a copy of $\mathcal{A}$. Every input from $\mathcal{E}$ is written on $\mathcal{A}$'s input tape, and conversely. Note that seeing `setup` inputs of given by $\mathcal{E}$ to each corrupted node $\mathcal{A}(X)$ allows $\mathcal{S}\mathtt{im}$ to learn the *src* and *dst* value of all corrupted nodes.

Simulating $\mathcal{F}_{\mathsf{link}}$ and $\mathcal{F}_{\mathsf{offline}}$: $\mathcal{S}\mathtt{im}$ internally runs these two ideal functionalities in a completely honest way. This provides $\mathcal{S}\mathtt{im}$ with valuable information. For instance, running $\mathcal{F}_{\mathsf{offline}}$ allows $\mathcal{S}\mathtt{im}$ to see the *ocomid* and key k.

Simulating key pairs: $\mathcal{S}\mathtt{im}$ must obtain a key pair for each honest node, but can not request $\mathcal{F}_{\text{reg}}$ for it, since, for an honest node X, $\mathcal{F}_{\text{reg}}$ only answers to the node itself. Consequently, $\mathcal{S}\mathtt{im}$ generates one fresh key pair for each honest node. Denote the key of $X \in \Omega_{\mathcal{S}\mathtt{im}}$ as $(pk_{\mathcal{S}\mathtt{im}(X)}, sk_{\mathcal{S}\mathtt{im}(X)})$. Additionally, $\mathcal{S}\mathtt{im}$ honestly relays $\mathcal{A}$'s queries to $\mathcal{F}_{\text{reg}}$ for the key pairs of corrupted nodes, and the corresponding answers. In this process, $\mathcal{S}\mathtt{im}$ in particular learns the public key of corrupted nodes.

Simulating topology dissemination: Topology dissemination must be performed through $\mathcal{F}$'s internally ran instance of $\mathcal{F}_{\mathsf{rtprop}}$. More exactly, route proposals are performed through $\mathcal{F}_{\mathsf{rtprop}}$, but the *route proposal policy* (*i.e.* the decision of

accepting routes, and about re-proposing them) is run by $\mathcal{F}$ and $\mathcal{A}$, for honest and corrupted nodes respectively. . Note that $\mathcal{F}$ is designed to allow $\mathcal{S}\texttt{im}$ to provide proposer inputs to $\mathcal{F}$, that relays it to the internally ran $\mathcal{F}_{\mathsf{rtprop}}$ instance. Thus, every time $\mathcal{A}$ want to provide a proposer input to $\mathcal{F}_{\mathsf{rtprop}}$ ion behalf of corrupted node X (as it would in a real execution), $\mathcal{S}\texttt{im}$ externally sends that input to $\mathcal{F}$. Conversely, when $\mathcal{F}$ sends back proposee inputs meant for corrupted node X, $\mathcal{S}\texttt{im}$ internally relays it to $\mathcal{A}$. Note that this relay position allows $\mathcal{S}\texttt{im}$ to perfectly know all portions of routes among corrupted nodes. In particular, this gives $\mathcal{S}\texttt{im}$ the abilities: to trace any portion of route made of corrupted nodes, given only a first hop (Y, cid), and to know the pseudonyms (and identity if corrupted) of the end-receivers. This is similar to the situation in proof of Theorem 2. Finally, during the topology dissemination phase, the $\mathcal{F}$ will leak information on route proposals (*i.e.* route-prop and subpath leaks from its internal $\mathcal{F}_{\mathsf{rtprop}}$ instance), which $\mathcal{S}\texttt{im}$ directly passes to $\mathcal{A}$.

However, throughout the above described process, in order to lay the groundwork for the simulation of oriented communications, $\mathcal{S}\texttt{im}$ applies slight modifications to proposee outputs. That is, it modifies the $c_{\texttt{one}X\to R}$ ciphertexts by an encryption of one under a public key controlled by $\mathcal{S}\texttt{im}$, so that, every time a link message arrives at an honest node, $\mathcal{S}\texttt{im}$ can decrypt the ciphertexts it carries. From the construction of the simulation in the proof of Theorem 2, it is clear that $\mathcal{S}\texttt{im}$ can proceed as follows: when $\mathcal{F}$ makes a $(\mathsf{proposee}, sid, PS_{X\to R}, c_{\texttt{one}X\to R}, (Y, cid))$ output to $\mathcal{A}(X)$, if $Y \in \Omega_{\mathcal{S}\texttt{im}}$, then it replaces $c_{\texttt{one}X\to R}$ with $\mathsf{Enc}(pk_{\mathcal{S}\texttt{im}(Y)}, 1)$; if $Y \in \Omega_c$, let the proposed route be $(Y, Z_1, \ldots, Z_n)$ with end-receiver Z_n, and Z_i be the first honest node on that route, then replace $c_{\texttt{one}X\to R}$ with $\mathsf{Enc}(pk_Y \cdot \cdots \cdot pk_{\mathcal{S}\texttt{im}(Z_i)}, 1)$; otherwise, if there are no honest node on the route, leave $c_{\texttt{one}X\to R}$ as is. Note that this approach requires that $\mathcal{S}\texttt{im}$ knows the public keys of corrupted nodes (that it learns by relaying $\mathcal{A}$'s requests to $\mathcal{F}_{\text{reg}}$). This replacement of encryptions of one is indistinguishable by the IK-CPA and USS properties of the Elgamal scheme.

Simulating intermediary corrupted sub-paths: Intermediary corrupted sub-paths are simulated exactly as in the proof of Theorem 2, in an independent and ordered manner. The details are thus not given here. The only difference being the `payload` messages that do not convey *rcid* value, and which simulation is actually simpler because no *return trip* is necessary.

Simulating payload messages from corrupted end-senders: After topology dissemination, nodes may be instructed at any moment to end-send messages. This is done on a per-message basis by $\mathcal{E}$. $\mathcal{S}\texttt{im}$ must simulate the sending, relaying, and receiving of payload messages, and the initialisation of oriented communications on behalf of honest nodes. We begin by describing the work of $\mathcal{S}\texttt{im}$ when a corrupted end-sender S is instructed to send a message by a $(\mathsf{send}, sid, ocomid, R, data)$ input. Recall that $\mathcal{S}\texttt{im}$ receives this input from $\mathcal{E}$ (and thus knows that *ocomid* is associated with end-sender S and end-receiver R). Upon such an input, $\mathcal{S}\texttt{im}$ passes it on to $\mathcal{A}$, and also immediately externally sends it to $\mathcal{F}(S)$. The simulation is

then divided into three major steps: **(I)** the simulation of the interaction between end-sender and end-receiver through $\mathcal{F}_{\mathsf{offline}}$, **(II)** the simulation of the *first leg* between end-sender and indirection node, and **(III)** the simulation of the *second leg* between indirection node and end-receiver.

Step (I) First of all, $\mathcal{A}(S)$, being semi-honest and thus following Π, will select a random routing table entry, in order choose an indirection node I. By assumption that end-sender and indirections nodes do not collude, this I is necessarily honest. Then, $\mathcal{A}(S)$ interacts with R through $\mathcal{F}_{\mathsf{offline}}$, as it would in a real execution. $\mathcal{S}\mathtt{im}$ sees and relays the input $(\mathsf{get}, sid, R, ocomid, c_{\mathtt{one}S\to I}, k)$ from $\mathcal{A}(S)$ to the simulated instance of $\mathcal{F}_{\mathsf{offline}}$. If R is corrupted, $\mathcal{A}(R)$ answers (still through the simulated $\mathcal{F}_{\mathsf{rtprop}}$ instance), and $\mathcal{S}\mathtt{im}$ has nothing to do. Otherwise, $\mathcal{S}\mathtt{im}$ must compute shares sh_1 and sh_2 to answer $\mathcal{A}(S)$. For that, $\mathcal{S}\mathtt{im}$ samples $sh_1, sh_2 \leftarrow_{\$} \mathbb{G}$, and answers with $(\mathsf{got}, sid, ocomid, sh_1, \mathsf{Enc}_{\mathsf{nopk}}(c_{\mathtt{one}S\to I}, sh_2)$.

Step (II) After the interaction through $\mathcal{F}_{\mathsf{offline}}$, $\mathcal{A}(S)$ and $\mathcal{S}\mathtt{im}(I)$ exchange several messages. More exactly, step (II) can be decomposed as: (II.i) $\mathcal{S}\mathtt{im}$ answering the $(\mathsf{sender\text{-}pick\text{-}route}, sid, ocomid, S)$ sent by $\mathcal{F}$, (II.ii) $\mathcal{A}(S)$ sending six messages as part of the oriented communication initialisation, (II.iii) $\mathcal{S}\mathtt{im}(I)$ answering with two `rtproprelay` messages, and (II.iv) $\mathcal{A}(S)$ finally sending back the `payload` message containing $PS_{I\to R}$ and then sending one or several `payload` messages with with $\{data\}_k$. Furthermore, in the simulation of the step (II) as a whole, we will assume that the route for the first leg, between $\mathcal{A}(S)$ and $\mathcal{S}\mathtt{im}(I)$, begins as $(\mathcal{A}(S) \overset{cid_1}{\longrightarrow} \mathcal{A}(Z_1) \overset{cid_2}{\longrightarrow} \mathcal{A}(\cdots) \overset{cid_i}{\longrightarrow} \mathcal{S}\mathtt{im}(Z_i))$. That is, it begins with a sequence of corrupted nodes, and necessarily hits a honest node Z_i at some point. Indeed, since I itself is honest, in the *worst case*, Z_i is actually I.

Step (II.i) Firstly, $\mathcal{S}\mathtt{im}$ must provide $\mathcal{F}$ with the first hop used by $\mathcal{A}(S)$. This is something $\mathcal{S}\mathtt{im}$ can learn. Indeed, $\mathcal{A}(S)$ will begin sending the first messages of the oriented communication initialisation, and at some point, $\mathcal{S}\mathtt{im}(Z_i)$ will internally receive messages of the form: $\langle \mathtt{flag} \| cid_i [\| rcid], c_1, c_2 \rangle$ from $\mathcal{A}(Z_{i-1})$. By construction of the topology dissemination simulation, c_1 and c_2 are encrypted solely under $pk_{\mathcal{S}\mathtt{im}(Z_i)}$. Thus, $\mathcal{S}\mathtt{im}$ can decrypt them, and in particular obtain $ocomid$, linking that message to the communication between $\mathcal{A}(S)$ and R. Furthermore, by knowing $\mathcal{A}(S)$ and $(\mathcal{A}(Z_{i-1}), cid_i)$, $\mathcal{S}\mathtt{im}$ can deduce the exact path taken by the message from $\mathcal{A}(S)$ to $\mathcal{S}\mathtt{im}(Z_i)$ (since it perfectly knows routes between corrupted nodes). From that, $\mathcal{S}\mathtt{im}$ deduces that $\mathcal{A}(S)$ used the first hop (Z_1, cid_1). Consequently, $\mathcal{S}\mathtt{im}$ sends $(\mathsf{sender\text{-}route}, sid, ocomid, S, (Z_1, cid_1))$ to $\mathcal{F}$. This ensures that the route(s) used by node S in (tine internally run instance of Π in) $\mathcal{F}$ are the same as those used by $\mathcal{A}(S)$.

Step (II.ii) The next step consists in simulating the relay of the first six messages from $\mathcal{A}(S)$ to I. This step is actually trivial: since I is honest, there are only intermediary corrupted sub-paths to simulate, which are handled as usual, in an independent manner. However, each message received by $\mathcal{S}\mathtt{im}(Z_i)$ from $\mathcal{A}(Z_{i-1})$, contains data encrypted under $pk_{\mathcal{S}\mathtt{im}(Z_i)}$ (which $\mathcal{S}\mathtt{im}$ can thus decrypt) that will

be useful for the simulation of the next step. More exactly, $\mathcal{S}\texttt{im}(Z_i)$ receives sh_2, pk^{ocom}, pk^{tmp}, $c_{sh_1}[0]$, and $c_{sh_1}[1]$, which $\mathcal{S}\texttt{im}$ stores along with $ocomid$.

Step (II.iii) In this third step, $\mathcal{S}\texttt{im}(Z_i)$ must send back two `rtproprelay` messages towards the end-sender $\mathcal{A}(S)$, *via* $\mathcal{A}(Z_{i-1})$. More accurately, after possibly several subpath leaks corresponding to intermediary corrupted sub-paths from the end-sender to the indirection node and back, $\mathcal{F}$ will leak (ocom-sender, sid, $(Z_i \overset{cid_{i-i}}{\longrightarrow} \dots \overset{cid_1}{\longrightarrow} S), ocomid \| cnt, {dst_R}^{src_I}$). At this point, $\mathcal{S}\texttt{im}$ retrieves the previously stored pk^{ocom} and pk^{tmp} associated to $ocomid$. Then, $\mathcal{S}\texttt{im}$ crafts a fresh encryption of ${dst_R}^{src_I}$ (learned in the leak) with pk^{ocom}, *i.e.* it sets $(c[0], c[1]) \leftarrow \mathsf{Enc}(pk^{ocom}, {dst_R}^{src_I})$. From this, $\mathcal{S}\texttt{im}(Z_i)$ sends the two following messages to $\mathcal{A}(Z_{i-1})$:

$$\Big\langle \texttt{rtproprelay} \| cid_i \| rcid, \mathsf{Enc}(pk^{tmp}, c[0]), \mathsf{Enc}(pk^{tmp}, 1) \Big\rangle$$

$$\Big\langle \texttt{rtproprelay} \| cid_i \| rcid', \mathsf{Enc}(pk^{tmp}, c[1]), \mathsf{Enc}(pk^{tmp}, 1) \Big\rangle$$

By construction of the simulation, these messages are ensured to be relayed (by corrupted nodes, played by $\mathcal{A}$), up to $\mathcal{A}(S)$.

Step (II.iv) This last step is simulated exactly as step (II.ii). That is, there are mainly intermediary corrupted sub-paths to simulate. However, in this step, for each each payload message sent by $\mathcal{A}(S)$ that contains actual data, $\mathcal{S}\texttt{im}$ gets $\{data\}_k$, along with the counter number cnt associated to each such piece of encrypted data. This knowledge will be used in the next simulation step.

Step (III) The simulation of the second leg, between indirection node and end-receiver, depends on whether the latter is corrupted or not. If R is honest, there are intermediary corrupted sub-paths to simulate, and at some point, $\mathcal{S}\texttt{im}$ will externally receive (continue?, sid, $ocomid \| cnt$) from $\mathcal{F}$ (on for each pieces of $data$ received by R in $\mathcal{F}$). $\mathcal{S}\texttt{im}$ accordingly externally sends (continue?, sid, $ocomid \| cnt$) to $\mathcal{F}$. This mechanism of continue exchange acts as a synchronisation point: $\mathcal{S}\texttt{im}$ answers back with a continue only when it has simulated all the other leaks from $\mathcal{F}$ that precede the continue? message. This ultimately ensures that $\mathcal{F}(R)$ makes its rcvd output only *after* all the intermediary sub-paths from I to R are simulated.

If R is corrupted, the situation is quite different. Assume, without loss of generality, that the route between I and R ends with $(\mathcal{S}\texttt{im}(Z_j) \overset{cid_{j+1}}{\longrightarrow} \mathcal{A}(Z_{j+1}) \overset{cid_{j+2}}{\longrightarrow} \mathcal{A}(\cdots) \overset{cid_n}{\longrightarrow} \mathcal{A}(R))$, *i.e.* that Z_j is the last honest node on the route. For each payload message received by R in (the internal Π instance within) $\mathcal{F}$, the latter will leak (ocom-rcvr, sid, $(Z_j \overset{cid_{j+1}}{\longrightarrow} \dots \overset{cid_n}{\longrightarrow} R), ocomid \| cnt, data$). This indicates to $\mathcal{S}\texttt{im}$ that it must *make* $\mathcal{S}\texttt{im}(Z_j)$ *deliver* $\{data\}_k$ to $\mathcal{A}(R)$ through $\mathcal{A}(Z_{j+1})$. Note however, that $\{data\}_k$ must be *exactly equal* to the SKE ciphertext that $\mathcal{A}(S)$ originally crafted (since, in a real execution, this SKE ciphertext stays the same from end-sender to indirection node to end-receiver). Therefore, here, the leaked cnt value is used as an *index* to retrieve $\{data\}_k$ learned during the simulation of step (II.iv). From this, $\mathcal{S}\texttt{im}(Z_j)$

sends the following message to $\mathcal{A}(Z_{j+1})$:

$$\langle \texttt{payload} \| cid_{j+1}, \mathsf{Enc}_{\texttt{nopk}}(c_{\texttt{one}}, \texttt{ocom} \| ocomid \| cnt), \mathsf{Enc}_{\texttt{nopk}}(c_{\texttt{one}}, \{data\}_k) \rangle$$

Here, $c_{\texttt{one}}$ is the adequate encryption associated to (Z_{j+1}, cid_{j+1}), which $\mathcal{S}\texttt{im}$ necessarily knows from the simulation of the topology dissemination phase. Also, because the exact same routes are constructed in the simulation and in $\mathcal{F}$, this message is ensured to go through $\mathcal{A}(Z_{j+1}), \mathcal{A}(Z_{j+2}), \ldots$, and to ultimately arrive at $\mathcal{A}(R)$ with ciphertexts solely encrypted under pk_R.

Indistinguishability: Indistinguishability stems from the IND-CPA, IK-CPA and USS properties, plus from the way $\mathcal{F}$ is constructed which allows $\mathcal{S}\texttt{im}$ to learn all the information it needs, an that allows $\mathcal{S}\texttt{im}$ to incorporate $\mathcal{A}$'s random choices in $\mathcal{F}$'s internal instance of Π (such as the route chosen by $\mathcal{A}(S)$).

More precisely, the simulation of this case differs from a real execution in the following points:

- The shares. The shares sh_1, sh_2 do not produce dst_R. But since I is honest by assumption, $\mathcal{A}$ only sees one share sh_1, and indistinguishability stems from the security of the secret sharing mechanism.
- If the route between $\mathcal{A}(S)$ and I contains at least one intermediary corrupted sub-path, then: (i) these nodes relay random data, and (ii) corrupted nodes in different intermediate sub-paths see ciphertext that are completely independent from each other. In particular, these ciphertexts are not a function of those sent by $\mathcal{A}(S)$. Indistinguishability here stems from the IND-CPA property for point (i), and from the USS property from point (ii).
- The point immediately above is also valid for the route between I and R, and the same arguments apply.
- Every time $\mathcal{S}\texttt{im}$ *delivers* messages to a corrupted node (that is, $\mathcal{A}(S)$ and possibly $\mathcal{A}(R)$), the latter receives ciphertexts *crafted* by $\mathcal{S}\texttt{im}(Z_k)$. Again, indistinguishability is ensured by the USS properties.

Simulating payload messages from honest end-senders: The simulation of honest end-senders actually follows the same general methodology as for corrupted end-senders: $\mathcal{S}\texttt{im}$ will simulate the intermediary sub-paths independently, and deliver messages to corrupted indirection nodes or end-receivers. Before all, note if neither I nor R is corrupted, there is nothing to simulate but intermediary corrupted sub-paths, and in particular, *ocomid* is never leaked to $\mathcal{S}\texttt{im}$.

The simulation of honest end-senders still features some important differences. Here, $\mathcal{S}\texttt{im}$ must simulate interactions with the end-receiver through $\mathcal{F}_{\mathsf{offline}}$, I may be corrupted, and the delivery of messages to corrupted indirection nodes or end-receivers necessitates new sources of information for $\mathcal{S}\texttt{im}$. We hereby assume that $\mathcal{F}(S)$ (for some $S \in \Omega_{\mathcal{S}\texttt{im}}$) receives input $(\mathsf{send}, sid, ocomid, R, data)$ from

$\mathcal{E}$. Three steps (I), (II) and (III) are identified, as for the case of corrupted end-senders.

Step (I) First of all, if R is corrupted, $\mathcal{F}$ leaks $(\mathsf{offline}, sid, ocomid, R)$ to $\mathcal{S}\mathtt{im}$. The latter thus knows it must simulate an interaction between *some* honest end-sender (whose identity it does not know) and node R through $\mathcal{F}_{\mathsf{offline}}$. For that, it uses the specified *ocomid*, a freshly generated symmetric key k, and encryptions of one under a key it controls, *i.e.* one it generates for the occasion (instead of an actual $c_{\mathrm{one}S\to I}$), so that it can decrypt and get the shares sh_1 and sh_2. It gives all that to $\mathcal{A}(R)$ in a `get` input. On the other hand, if R is honest, $\mathcal{S}\mathtt{im}$ receives nothing from $\mathcal{F}$, and does not need to simulate anything. It still generates $sh_1, sh_2 \leftarrow_{\$} \mathbb{G}$ for later use.

Step (II) After this, $\mathcal{S}\mathtt{im}$ must simulate the first leg. Depending on the indirection node and on the route chosen by $\mathcal{F}(S)$, the latter may or may not contain intermediary corrupted sub-paths. If so, they are handled as usual, on the way forward and back. Also, the indirection node may or may not be corrupted, depending on the random choice of $\mathcal{F}(S)$. If I is honest, then there is nothing else to simulate for the first leg.

If I is corrupted, $\mathcal{F}$ leaks $(\mathsf{ocom\text{-}I}, sid, (Z_j \overset{cid_{j+1}}{\to} \ldots \overset{cid_n}{\to} I), \mathtt{flag}, ocomid \| cnt)$ for each of the six first messages that are part of the oriented communication initialisation. Each of these leaks indicate that $\mathcal{S}\mathtt{im}$ must make $\mathcal{S}\mathtt{im}(Z_j)$ *deliver* a message containing either $sh_2, pk^{ocom}, pk^{tmp}, c_{sh_1}[0]$, or $c_{sh_1}[1]$ to $\mathcal{A}(I)$ through $\mathcal{A}(Z_{j+1})$. Here again, it is the value of *cnt* of the leak lets $\mathcal{S}\mathtt{im}$ know which particular piece of information must be sent. However, $\mathcal{S}\mathtt{im}$ must generate $sh_2, pk^{ocom}, pk^{tmp}, c_{sh_1}[0]$, and $c_{sh_1}[1]$ by itself. For that, it uses the following values: freshly generated key pairs (pk^{tmp}, sk^{tmp}) and (pk^{ocom}, sk^{ocom}); sh_1, sh_2 as obtained from step (I); and $c \leftarrow \mathsf{Enc}(pk^{ocom}, sh_1)$. From this, messages are *crafted and delivered* by $\mathcal{S}\mathtt{im}(Z_j)$ to $\mathcal{A}(I)$ in the usual way. Then $\mathcal{A}(I)$ processes the messages, and answers with two `rtproprelay` messages, that $\mathcal{S}\mathtt{im}(Z_j)$ receives, decrypts, and recognises as belonging to the session identified by *ocomid*. These messages are however simply discarded. More intermediary sub-paths are possibly simulated, and at some point, when in (the internal Π instance within) $\mathcal{F}$, $\mathcal{F}(I)$ receives the `payload` message containing the pseudonym, $\mathcal{F}$ makes a $\mathsf{ocom\text{-}I}$ leak specifying the pseudonym $PS_{I\to R}$. With this value, $\mathcal{S}\mathtt{im}$ can thus deliver the pseudonym to $\mathcal{A}(I)$ in an adequately crafted ciphertext. For each following `payload` message containing some *data*:

- Either R is corrupted as well, and $(\mathsf{ocom\text{-}I}, sid, (Z_j \overset{cid_{j+1}}{\to} \ldots \overset{cid_n}{\to} I), ocomid \| cnt,$ $data))$ is leaked, specifying *data*. $\mathcal{S}\mathtt{im}$ can thus encrypt *data* with k (the key generated in step (I)), and make $\mathcal{S}\mathtt{im}(Z_j)$ deliver it to $\mathcal{A}(I)$.
- Either R is honest, and $\mathcal{S}\mathtt{im}$ delivers $\{data'\}_{k'}$ for random values $data'$, $k' \leftarrow_{\$} \{0,1\}^*$. If there are multiple *data* `payload` messages to deliver, $\mathcal{S}\mathtt{im}$ uses the same key k' for all of them (but a different $data'$).

Step (III) For the second leg, we distinguish five corruption configurations: (a) I and R are both honest, (b) R is corrupted, and (c) I is corrupted, (d) both I and R are corrupted *but* there is at least one honest node on the route between them, and (e) both I and R are corrupted, and there are only corrupted nodes on the route.

In case (a), there is nothing else to simulate but intermediary corrupted sub-paths. In case (b), for each payload message, $\mathcal{S}\texttt{im}$ must deliver $\{data\}_k$ to $\mathcal{A}(R)$ when $\mathcal{F}$ leaks $(\mathsf{ocom\text{-}rcvr}, sid, (Z_j \overset{cid_{j+1}}{\rightarrow} \ldots \overset{cid_v}{\rightarrow} R), ocomid \| cnt, data)$. For that, $\mathcal{S}\texttt{im}$ generates $\{data\}_k$ using $data$ from the leak and k from step (I), and makes $\mathcal{S}\texttt{im}(Z_j)$ deliver it to $\mathcal{A}(R)$ as usual. In case (c), there is no need to deliver anything to R, since it is honest. However, since I is corrupted, $\mathcal{S}\texttt{im}$ will externally receive $(\mathsf{I\text{-}pick\text{-}route}, sid, ocomid, I)$ from $\mathcal{F}$. $\mathcal{S}\texttt{im}$ must answer with the first hop chosen by $\mathcal{A}$, which it can learn in the same manner as described previously (step (II.i) of the simulation of corrupted end-senders). $\mathcal{S}\texttt{im}$ thus sends back $(\mathsf{I\text{-}route}, sid, ocomid, I, (Y, cid))$ to $\mathcal{F}$.

This latter I-pick-route/I-route exchange must also be carried out in cases (d) and (e). Note that in case (e), there is no honest node on the route for $\mathcal{S}\texttt{im}$ to *intercept* the message, but $\mathcal{S}\texttt{im}$ can trivially learn the first hop used by $\mathcal{A}(I)$ by the $(\mathsf{ocom\text{-}rcvr}, sid, (I \overset{cid_1}{\rightarrow} \ldots \overset{cid_v}{\rightarrow} R), ocomid \| cnt, data)$ leak from $\mathcal{F}$, that specifies the full route from I to R. For case (e), this I-pick-route/I-route exchange is the only element to simulate for $\mathcal{S}\texttt{im}$: since the whole route is corrupted, it is $\mathcal{A}$ that performs all the actions. In case (d), there are possibly intermediary corrupted sub-paths, but most importantly, using the last honest node on the route, $\mathcal{S}\texttt{im}$ must deliver to $\mathcal{A}(R)$ what the *first* honest node on the route receives from $\mathcal{A}(I)$. This is performed by $\mathcal{S}\texttt{im}$ for each payload message in the oriented communication session, using techniques already described.

Indistinguishability: By the same arguments as for the case of corrupted end-senders, the simulation of intermediary corrupted sub-paths, and the delivery of messages to corrupted nodes, are indistinguishable from a real execution. Other points that needs to be mentioned are:

- Public keys pk^{tmp} and pk^{ocom} are generated by $\mathcal{S}\texttt{im}$, instead of the actual end-sender S. However, this perfectly simulates a real execution, since $\mathcal{S}\texttt{im}$ generates these keys exactly as an honest S would do.
- The encryption of $data$. When the end-receiver is honest but the indirection node is not, $\mathcal{S}\texttt{im}$ delivers $\{data'\}_{k'}$ for a random $data'$ and k' instead of the actual data and key. This is indistinguishable, by the IND-CPA security of AES, and by the randomness of the key that the end-sender and end-receiver agreed on. When both the end-receiver and indirection node are corrupted, $\mathcal{S}\texttt{im}$ generates $\{data\}_k$ itself (this perfectly simulates how an actual honest end-sender would behave). But note that care is taken to give the *exact same* ciphertext $\{data\}_k$ both to $\mathcal{A}(I)$ and $\mathcal{A}(R)$. This simulation is thus

indistinguishable from a real execution, where $\mathcal{A}(I)$ and $\mathcal{A}(R)$ indeed expect to see the same SKE ciphertext. Lastly, when the end-receiver is corrupted but the indirection node is not, $\mathcal{S}\texttt{im}$ generates $\{data\}_k$ just before delivering it to $\mathcal{A}(R)$. This yields a simulation indistinguishable form a real execution, since no corrupted node get to see $\{data\}_k$ other than R.

– When R is corrupted, $\mathcal{S}\texttt{im}$ interacts with it through $\mathcal{F}_{\mathsf{offline}}$ on behalf on the honest end-sender S. Although $\mathcal{S}\texttt{im}$ does not know the actual identity of this end-sender, note that, by construction of $\mathcal{F}_{\mathsf{offline}}$, in a real execution, $\mathcal{A}(R)$ does noes expect to get any information on the identity of the end-sender. Thus, the fact that $\mathcal{S}\texttt{im}$ does not know the identity of the end-sender is not an issue. Secondly, although $\mathcal{S}\texttt{im}$ does not use an actual encryption of one $c_{\texttt{one}S\to I}$ existing in the network, but $c_{\texttt{one}} \leftarrow \mathsf{Enc}(pk, 1)$ for a freshly generated pk. However, by the IK-CPA and USS properties, $c_{\texttt{one}}$ are indistinguishable from an actual $c_{\texttt{one}S\to I}$ that $\mathcal{A}(R)$ would expect to receive in a real execution.

– The shares: when R is honest, $\mathcal{S}\texttt{im}$ generates the shares randomly. Since $\mathcal{A}$ only sees at most one of these shares shares (namely, sh_2, *via* I when it is corrupted) this element of the simulation is indistinguishable from a real execution by the security of the secret sharing mechanism.

This terminates the description of the simulator. The conclusion of the proofs goes as for the proof of Theorem 3. □

B.6. Analysis of $\mathcal{F}$

This section provides the proofs of the two theorems from Section 5.6.3 (pages 131 and 133). The first proof deals with SA, RA and SU, and the second one with MU.

B.6.1. Proof of Theorem 5 (SA, RA, SU)

Proof of Theorem 5. Following the proof methodology of Backes *et al.* [Bac+13], to simplify the proof, the challenger and adversary $\mathcal{A}'(\mathcal{A})$ are first modeled into deterministic machines by explicitly proving them random strings r_{Ch} and $r_{\mathcal{A}}$ as input. Fixing these random strings in turn fixes all routes created and used in the network, the choices of indirection nodes. It also fixes all the set of corrupted nodes, the send inputs created by $\mathcal{A}$, the *src* and *dst* value of each node. More generally, this makes the algorithm $\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha,b,r_{Ch})}$ completely deterministic. That is, the execution of $\mathcal{F}$ by the challenger is fixed $r_{\mathcal{A}}$ and r_{Ch}. Therefore, the algorithms $\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha_{RA},0,r_{Ch})}$ and $\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha_{RA},1,r_{Ch})}$ differ only by the value of the bit b, which means that the difference in what $\mathcal{A}$ learns from their execution solely depends on the challenge session(s). This allows us to focus only on what the adversary learns from the challenge (*i.e.* on the difference of the leaks that $\mathcal{A}$ gets from $\mathcal{F}$ when $b = 1$ and when $b = 0$).

In the analysis, we consider the following *distinguishing events*. These events are defined for fixed values for $r_{Ch}, r_{\mathcal{A}} \in \{0,1\}^*$ and $b \in \{0,1\}$, since this triplet of values fully determines whether the event happens or not.

$E_{SA}(r_{Ch}, r_{\mathcal{A}}, b) \coloneqq$ there is at least one corrupted node on the chosen route between the end-sender S_b and the indirection node (included).

$E_{RA}(r_{Ch}, r_{\mathcal{A}}, b) \coloneqq$ there is at least one corrupted node on the chosen route between the indirection node (included) and the end-receiver R_b.

$E_{SU}(r_{Ch}, r_{\mathcal{A}}, b) \coloneqq$ there is at least one corrupted node on the *full* chosen route between the end-sender S_a and the end-receiver R_a and one corrupted node on the *full* chosen route between the end-sender $S_{a'}$ and the end-receiver $R_{a'}$.

Given these events, we show that, for any $r_{Ch}, r_{\mathcal{A}} \leftarrow\!\!\$\ \{0,1\}^*$ and for $\texttt{PROP} \in \{SA, RA, SU\}$:

$$\begin{aligned} &\Pr\left[\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F}, \alpha_{\texttt{PROP}}, 0, r_{Ch})} = 0 \;\middle|\; \neg E_{\texttt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)\right] \\ =&\Pr\left[\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F}, \alpha_{\texttt{PROP}}, 1, r_{Ch})} = 0 \;\middle|\; \neg E_{\texttt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 1)\right] + \mathsf{negl}(\lambda) \end{aligned} \tag{B.1}$$

Intuitively, this holds because, when $b = 0$ and $\neg E_{\texttt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)$, and when $b = 1$ and $\neg E_{\texttt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 1)$, $\mathcal{A}$ gets the same information from $\mathcal{F}$, up to *e.g.* a pseudonym, which does not allow her to distinguish the two case with more than a negligible advantage.

More accurately, for the RA challenge (the most complex case of the proof), when E_{RA} does *not* happen, $\mathcal{A}$ gets:

- proposer, proposee, rtprop, subpath leaks during topology dissemination. These leaks do not give information on R_0 nor R_1 *as end-receivers* by the indistinguishability of pseudonyms and the IK-CPA property of the Elgamal scheme. Thus, in this phase, $\mathcal{A}$ learns no information that could help her towards winning the challenge.
- For each oriented communications *except* the challenge one, $\mathcal{A}$ gets send, rcvd, l-pick-route, l-route, sender-pick-route, sender-route, ocom-l, ocom-sender, ocom-rcvr, and offline leaks. The only information leaked to $\mathcal{A}$ that relates to R_0 or R_1 are the ${dst_R}^{src_I}$ values that the ocom-sender leaks contain. These leaks happen when the sender of the oriented communication is corrupted. Recall that, by assumption, in this case, the indirection node is honest. Thus, $\mathcal{A}$ receives ${dst_{R_b}}^{src_I}$ values, for unknown src_I values. Thus, given polynomially many ${dst_{R_b}}^{src_I}$ values, $\mathcal{A}$ can not learn information on dst_{R_0} nor dst_{R_1}.
- For the challenge communication between S and R_b, recall that, by the definition of the function α_{RA}, the end-sender S in the challenge session is assumed honest (since it does not make sense to check the anonymity of the end-receiver w.r.t. the end-sender, which explicitly knows its identity). Also, $\mathcal{A}$ does not get any information on the second leg, since when $\neg E_{RA}(r_{Ch}, r_{\mathcal{A}}, b)$ happens, the entire second leg, including the indirection node, is honest. In definitive, $\mathcal{A}$ gets only subpaths leaks on the first leg. These leaks do not give any direct or indirect information on R_0 nor R_1.

For a fixed random string r_{Ch}, these three sets of information that $\mathcal{A}$ learns during the RA challenge are actually the same whether $b = 0$ or $b = 1$. Therefore, if $R_{\mathcal{A}} \subseteq \{0,1\}^*$ is the

subset of random strings $r_{\mathcal{A}}$ such that $\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha_{RA},0,r_{Ch})} = 0$ when $\neg E(r_{Ch}, r_{\mathcal{A}}, 0)$, then because $\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})$ is a deterministic machine $\forall\ r_{\mathcal{A}} \in R_{\mathcal{A}}$, $\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha_{RA},1,r_{Ch})} = 0$ when $\neg E(r_{Ch}, r_{\mathcal{A}}, 1)$ except with negligible probability (that depends polynomially on $\mathsf{Adv}^{\mathsf{ps-ind}}(\lambda) + \mathsf{Adv}^{\mathsf{ik\text{-}cpa}}(\lambda)$). The probabilities being the same for every random string r_{Ch}, by the law of total probability, eq (B.1) holds.

Then, for the case of SA and SU, the situation is simpler. In the SA challenge, when E_{SA} does not happen, $\mathcal{A}$ gets at most the following leaks about the challenge communication between S_b and R (when R, in particular, is corrupted): $(\mathsf{offline}, sid, ocomid, R)$, $(sid, R, ocomid, c_{\mathsf{one}_{A\to I}}, k)$, and $(\mathsf{ocom\text{-}rcvr}, sid, (Z_j \xrightarrow{cid_{j+1}} \dots \xrightarrow{cid_n} R), ocomid \| cnt, data)$. $\mathcal{A}$ also gets $\mathsf{subpaths}$ leaks relating to the second leg. $\mathcal{A}$ however does no get any information on the first leg, since when $\neg E_{SA}(r_{Ch}, r_{\mathcal{A}}, b)$ happens, the entire first leg, including the indirection node, is honest. As a result, $\mathcal{A}$ get no direct or indirect information relating to S_b, and, by an argument similar to the case of RA, the equality (B.1) can be show to hold perfectly for SA(*i.e.* with a negligible factor of zero). Likewise, for SU, when $\neg E_{SU}$, then the entire routes between S_a and R_a, and $S_{a'}$ and $R_{a'}$ consist only in honest nodes. Thus $\mathcal{A}$ gets *no information at all* on the two these communications (*i.e.* no information relating to the challenge), which trivially yields the result.

Using equality (B.1), we show that, for any $r_{Ch}, r_{\mathcal{A}} \leftarrow\!\!\$\ \{0,1\}^*$ and $\mathtt{PROP} \in \{SA, RA, SU\}$, the following holds:

$$
\begin{aligned}
&\Pr\left[\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha_{\mathtt{PROP}},0,r_{Ch})} = 0\right] \\
&= \Pr\left[\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha_{\mathtt{PROP}},0,r_{Ch})} = 0 \;\middle|\; E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)\right] \cdot \Pr[E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)] \\
&\quad + \Pr\left[\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha_{\mathtt{PROP}},0,r_{Ch})} = 0 \;\middle|\; \neg E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)\right] \cdot \Pr[\neg E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)] \\
&= \Pr\left[\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha_{\mathtt{PROP}},0,r_{Ch})} = 0 \;\middle|\; E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)\right] \cdot \Pr[E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)] \\
&\quad + (\Pr\left[\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha_{\mathtt{PROP}},1,r_{Ch})} = 0 \;\middle|\; \neg E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 1)\right] + \mathsf{negl}(\lambda)) \cdot \Pr[\neg E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)] \\
&\leq \Pr[E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)] \\
&\quad + (\Pr\left[\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha_{\mathtt{PROP}},1,r_{Ch})} = 0 \;\middle|\; \neg E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 1)\right] + \mathsf{negl}(\lambda)) \cdot \Pr[\neg E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)] \\
&\leq \Pr[E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)] + \Pr\left[\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha_{\mathtt{PROP}},1,r_{Ch})} = 0 \;\middle|\; \neg E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 1)\right] + \mathsf{negl}(\lambda) \\
&\leq \Pr[E_{\mathtt{PROP}}(r_{Ch}, r_{\mathcal{A}}, 0)] + \Pr\left[\mathcal{A}'(\mathcal{A}, r_{\mathcal{A}})^{Ch(\mathcal{F},\alpha_{\mathtt{PROP}},1,r_{Ch})} = 0\right] + \mathsf{negl}(\lambda)
\end{aligned}
$$

We have almost completed the proof of the theorem. Remains only to to quantify the probability of E_{SA}, E_{RA} and E_{SU}. Whether $b = 0$ or $b = 1$, the probabilities of these events for any fixed $r_{Ch}, r_{\mathcal{A}} \in \{0,1\}^*$ is bounded above as follows. The probabilities are

taken on the random choice of $\mathcal{A}$ in corrupting a specific subset of nodes.

$$\begin{aligned}
\Pr[E_{SA}(r_{Ch}, r_{\mathcal{A}}, b)] &= 1 - \Pr[\text{no node on the route from } S_b \text{ to } I \text{ is corrupted}] \\
&= 1 - \frac{\binom{|\Omega|-|rt_b|}{|\Omega_c|}}{\binom{|\Omega|}{|\Omega_c|}} \\
&\leq 1 - \frac{\binom{|\Omega|-l_{max}}{|\Omega_c|}}{\binom{|\Omega|}{|\Omega_c|}} \\
\Pr[E_{RA}(r_{Ch}, r_{\mathcal{A}}, b)] &\leq 1 - \frac{\binom{|\Omega|-l_{max}}{|\Omega_c|}}{\binom{|\Omega|}{|\Omega_c|}} \text{ (similarly to SA)} \\
\Pr[E_{SA}(r_{Ch}, r_{\mathcal{A}}, b)] &= 1 - \Pr\begin{bmatrix}\text{no node between } S_a \text{ and } R_a \text{ is corrupted} \vee \\ \text{no node between } S_{a'} \text{ and } R_{a'} \text{ is corrupted}\end{bmatrix} \\
&\leq 1 - \Pr[\text{no node between } S_a \text{ and } R_a \text{ is corrupted}] \\
&\leq 1 - \frac{\binom{|\Omega|-2l_{max}+1}{|\Omega_c|}}{\binom{|\Omega|}{|\Omega_c|}}
\end{aligned}$$

Since the whole proof holds for any randomly sampled $(r_{Ch}, r_{\mathcal{A}})$, we can remove the explicit random strings (making $\mathcal{A}'(\mathcal{A})$ and Ch probabilistic again), and obtain the theorem.

□

B.6.2. Proof of Theorem 6 (MU-tracing)

This section provides the full proof of Theorem 6, stated on page 133.

Proof of Theorem 6. Following Definition 26, let us first study the case when nodes $\mathcal{Z}_1, \ldots, \mathcal{Z}_K$ are on the first leg. For a given $i \in I_{\mathcal{F}}^{MU\text{-}R}$, $view_b(i)$ is:

$$\mathsf{Context}_{R_0,R_1}^{\mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K}(i) =$$
$$\begin{cases}
\{(Z, src_Z, dst_Z, (pk_Z, sk_Z)) \mid Z \in \Omega_c\} \\
\cup \{(Z, R', PS_{Z \to R'}, c_{\mathsf{one}\,Z \to R'}, cid_{Z \to R'}) \mid Z \in \Omega_c, R' \in \Omega \setminus \{R_0, R_1\}\} \\
\cup \ \{(Z, (N, PS_{Z \to R_a}, c_{\mathsf{one}\,Z \to R_a}, cid_{Z \to R_a})) \mid a \in \{0,1\}, Z \in \Omega_c \setminus (\mathcal{Z}_1 \cup \ldots \mathcal{Z}_K)\}
\end{cases}$$

$$\mathsf{View}^{\Omega_c \setminus (\mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K)}_{S_a \to R_a, ocomid_a, init}(i) =$$

$$\begin{cases}
(\mathsf{subpath}, sid, sp_1), \ldots, \\
\forall Z' \in (nodes(sp_i)) \cap \Omega_c, (Z', PS_{Z' \to I_a}, c_{\mathsf{one}\, Z' \to I_a}) \\
\text{if } S_a \in \Omega_c, \\
\quad (\mathsf{send}, sid, ocomid_a, R_a, data_a), \forall data_a \in \{data_a\} \\
\quad (\mathsf{sender\text{-}pick\text{-}route}, sid, ocomid_a, S_a) \\
\quad (\mathsf{sender\text{-}route}, sid, ocomid_a, (Z'_{k,i_k+1}, cid_{k,i_k+1})), (Z'_{k,i_k+1} \overset{cid_{k,i_k+2}}{\longrightarrow} \cdots \overset{cid_{k,n_k}}{\longrightarrow} Z'_{k,n_k}) \\
\quad \forall Z' \in \{Z'_{k,1}, \ldots, Z'_{k,i_k}\} \cap \Omega_c, (Z', PS_{Z' \to I_a}, c_{\mathsf{one}\, Z' \to I_a}) \\
\quad (\mathsf{ocom\text{-}sender}, sid, (Z'_{k,1} \overset{cid_{k,2}}{\longrightarrow} \cdots \overset{cid_{k,i_k}}{\longrightarrow} S_a), \mathtt{payload}, ocomid_a \| cnt, dst_{R_a}{}^{src_{I_a}}) \\
\text{if } I_a \in \Omega_c, \\
\quad (\mathsf{ocom\text{-}l}, sid, (Z'_{k,1} \overset{cid_{k,2}}{\longrightarrow} \cdots \overset{cid_{k,i_k}}{\longrightarrow} I_a), \mathtt{flag}, ocomid_a \| cnt) \text{ for } cnt \in \{1, \ldots, 5, 8\} \\
\quad \forall Z' \in \{Z'_{k,1}, \ldots, Z'_{k,i_k}\} \cap \Omega_c, (Z', PS_{Z' \to I_a}, c_{\mathsf{one}\, Z' \to I_a}) \\
\quad PS_{I_a \to R_a} \\
\quad (\mathsf{l\text{-}pick\text{-}route}, sid, ocomid_a, I_a) \\
\quad (\mathsf{l\text{-}route}, sid, ocomid_a, (Z'_{k,i_k+1}, cid_{k,i_k+1})), (Z'_{k,i_k+1} \overset{cid_{k,i_k+2}}{\longrightarrow} \cdots \overset{cid_{k,n_k}}{\longrightarrow} Z'_{k,n_k}) \\
\quad \forall Z' \in \{Z'_{k,i_k+1}, \ldots, Z'_{k,n_k}\} \cap \Omega_c, (Z', PS_{Z' \to R_a}, c_{\mathsf{one}\, Z' \to R_a}) \\
\quad (\mathsf{ocom\text{-}l}, sid, (Z'_{k,1} \overset{cid_{k,2}}{\longrightarrow} \cdots \overset{cid_{k,i_k}}{\longrightarrow} I_a), \mathtt{payload}, ocomid_a \| cnt) \text{ for } cnt \in [9, |\{data_a\}| + 9] \\
\quad \text{where all node } Z' \text{ in this view does } \textit{not} \text{ belong to the set } \mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K
\end{cases}$$

$$\mathsf{View}^{\mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K}_{S_{a'} \to R_{a'}, ocomid_{a'}}(i) =$$

$$\begin{cases}
(\mathsf{subpath}, sid, sp_1 = (Z_{1,1} \overset{\mathbf{cid_{1,2}}}{\longrightarrow} \cdots \overset{\mathbf{cid_{1,n_1}}}{\longrightarrow} Z_{1,n_1}), \mathtt{flag}), \ldots, (\mathsf{subpath}, sid, sp_K), \\
\forall Z \in (\mathcal{Z}_1 \cup \ldots \mathcal{Z}_K) \cap \Omega_c, (Z, \mathbf{PS_{Z \to R_{a'}}}, \mathbf{c}_{\mathsf{one}\, \mathbf{Z \to R_{a'}}})
\end{cases}$$

We reason on the information in the commit view, and study if it may (indirectly) allow the adversary to link the information in the challenge view to R_0 or R_1. More exactly, we have to analyse whether the adversary can know of R_a in the commit view (the first view) is the same as $R_{a'}$ in the challenge view (the second view). We show that it is not the case, *i.e.* that $\mathcal{A}$ can not distinguish between $view_0(i)$ and $view_1(i)$.

Both in the commit and challenge view, $\mathcal{A}$ gets leaks that depend on R_0 and/or R_1, under the form of *dst* values in pseudonyms, or of public keys in encryptions of one. We show, however, that the information on R_a in the commit view can not be used to by the adversary to distinguish with more than a negligible advantage whether the information in the challenge view relates to the same R_a or to a different $R_{a'}$. This holds by the by the indistinguishability of pseudonyms and the IK-CPA property, and because the definition of the MU-tracing property ensures that nodes which appear in the commit view do not appear in the challenge one (indeed, the commit view is taken over the set of nodes $\mathcal{N} \coloneqq \Omega_c \setminus (\mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K)$).

– In the commit view, even if S and I_a are both honest, $\mathcal{A}$ gets sub-paths and tuples $(Z', PS_{Z' \to I_a}, c_{\mathsf{one}\, Z' \to I_a})$. Thanks to the provided $\mathsf{Context}$, $\mathcal{A}$ directly knows that

these pseudonyms and encryption relate to I_a. But that does not say anything on R_a, and can not be used by $\mathcal{A}$ to win the challenge. Also, note that the set of nodes (and thus all sub-paths) that appear in the commit view, and the set of nodes (and sub-paths) that appear in the challenge view are disjoint. Therefore, in themselves, sub-paths do not provide any advantage to the adversary towards winning the challenge.

- If I_a is corrupted (and thus by assumption S_a is honest), $\mathcal{A}$ additionally gets $PS_{I_a \to R_a}$ and several tuples $(Z', PS_{Z' \to R_a}, c_{\texttt{one}Z' \to R_a})$ in the commit view. Then, in the challenge view, $\mathcal{A}$ gets tuples $(Z, PS_{Z \to R_a}, c_{\texttt{one}Z \to R_a})$, for *different* nodes $Z \neq Z'$. If there exists an adversary capable of distinguishing $view_0(i)$ and $view_1(i)$ based on these information, it is possible to construct an adversary $\mathcal{B}$ that breaks the indistinguishability of pseudonyms or the IK-CPA property. The reduction is similar to that of proof of Theorem 3 in Section B.4 of this appendix: since there are at most l_{max} challenge pseudonyms and encryptions of one in the challenge view, it can be shown that $\delta_R \leq l_{max} \cdot (\mathsf{Adv}^{\mathsf{ps-ind}}_{\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{ik\text{-}cpa}}_{\mathcal{A}}(\lambda))$ by a hybrid argument
- If S_a is corrupted (and thus, by assumption I_a is honest), $\mathcal{A}$ gets tuples $(Z', PS_{Z' \to I_a}, c_{\texttt{one}Z' \to I_a})$ in the commit view, as well as the identity of R_a. $\mathcal{A}$ thus *deduces* the value of a, since it knows the identities of R_0 and R_1. The latter fact does not help $\mathcal{A}$ at all towards winning the challenge. And, by an argument similar to the above point, even if $I_a = R_0$ or R_1 (which happens with probability $2/|\Omega|$, the tuples $(Z', PS_{Z' \to I_a}, c_{\texttt{one}Z' \to I_a})$ only provide a negligible advantage to adversary.

As a result, we have that $\delta_R = \mathsf{negl}(\lambda)$, and MU-tracing holds with all but a negligible advantage for the second leg.

The case of MU-tracing on the first leg is similar, but features one difference. In the challenge view, $\mathcal{A}$ gets one or more pseudonyms $PS_{Z \to I_{a'}}$, where Z is a corrupted node in the set $\mathcal{Z}_1 \cup \cdots \cup \mathcal{Z}_K$, and $I_{a'}$ is either I_0 or I_1. If $\mathcal{A}$ can determine whether this $I_{a'}$ in the challenge session is the same as the node I_a in the commit view, then she wins the challenge. When the indirections nodes I_0 and I_1 are both honest, then the same arguments as for the case of the second leg apply, and it is possible to show that the adversary has a negligible advantage in distinguishing $view_0(i)$ from $view_1(i)$.

However, when I_0 or I_1, or both, are corrupted the adversary wins with probability one, independently of the values of a and a'. We prove that by a case study of corruptions states of corrupted nodes, and on whether $a = a'$ or $a \neq a'$. Firstly, let us assume that $a = a'$ and that $I_a = I_{a'}$ is corrupted (the corruption state of I_{1-a} does not matter). Then, in the commit view, $\mathcal{A}$ recognises $PS_{Z' \to I_a}$ as the pseudonym of the corrupted node Z' towards the corrupted indirection node I_a, and trivially knowns the identity of I_a. Likewise, in the challenge session, $\mathcal{A}$ recognises $PS_{Z \to I_{a'}}$ as the pseudonym of the corrupted node Z towards $I_{a'}$, and trivially knows the identity of $I_{a'}$. Thus, $\mathcal{A}$ can observe that $I_a = I_{a'}$, and guess that she is observing $view_0(i)$ with probability one. Secondly, assume that $a \neq a'$, and that I_0 and I_1 are both corrupted. Then, by a similar argument

as the first case, $\mathcal{A}$ can deduce the identities of the indirections nodes in the commit and challenge view, observe that they are different, and deduce that $b = 1$. Finally, when $a \neq a'$ and only I_a or $I_{a'}$ is corrupted (but not both), $\mathcal{A}$ can only deduce the identity of the indirection node of the commit view, or of the challenge view (respectively). Since this case only arises in this particular configuration, it *indirectly* indicates to $\mathcal{A}$ that she is witnessing $view_1(i)$.

The proof for the first leg can be concluded in the same manner as proof of Theorem 5, by introducing a *distinguishing event* $E_{MU\text{-}S}$ defined as:

$$\begin{aligned}\Pr[E_{MU\text{-}S}] &= \Pr[I_0 \in \Omega_c \vee I_1 \in \Omega_c] \\ &= 1 - \Pr[I_0, I_1 \in \Omega] \\ &\leq 1 - \frac{\binom{|\Omega|-2}{|\Omega_c|}}{\binom{|\Omega|}{|\Omega_c|}}\end{aligned}$$

An analysis based on this distinguishing event allows to obtain the theorem, stating that:

$$\delta_S \leq 1 - \frac{\binom{|\Omega|-2}{|\Omega_c|}}{\binom{|\Omega|}{|\Omega_c|}} + \mathsf{negl}(\lambda)$$

□

B.7. Towards UC-realising $\mathcal{F}_{\mathsf{link}}$

The proof that the proposed protocol UC-realises $\mathcal{F}$ (in particular) lies in the $\mathcal{F}_{\mathsf{link}}$-hybrid protocol, which in the UC-framework can be identified as an assumption. Namely, $\mathcal{F}_{\mathsf{link}}$ represents the assumption that the controlled traffic rate, the dummy messages, and the message re-ordering mechanisms prevent external adversaries from even observing the exchanged messages between neighboring nodes (except maybe with a negligible advantage). This section is about breaking down and studying the feasibility and practicality of this assumption. For that, we describe the protocol Π_{link}, and present a slightly modified version of $\mathcal{F}_{\mathsf{link}}$ compared to the one defined in Chapter 5 (p. 106). Then, we put forwards a (partial) proof that Π_{link} UC-realises that variant of $\mathcal{F}_{\mathsf{link}}$.

B.7.1. Modeling Π_{link}

The protocol Π_{link} is presented in Fig. B.3. It is written with the same formalism as the other protocols presented in Chapter 5, and from the point of view of one node X. There are some points of Π_{link} that need to be explained. First, note that according to Π_{link}, the first action of any node X is to perform a DHKA with each of its neighbors. Then, Π_{link} (just as $\mathcal{F}_{\mathsf{link}}$) takes link-send inputs, and makes link-rcvd outputs. When X gets a link-send input (line 10), it simply places the input message in the corresponding neighbor pool. When X receives a message from a neighbor (line 12), it decrypts the header with the link keys from DHKAs, and checks whether it is a real or dummy message. Dummy

messages are simply discarded, while real messages are output (to the parent ITI, *e.g.* $\mathcal{E}$ or an instance of Π). In both cases, X also updates its incoming traffic counters $n_{I_{dum}}$ and $n_{I_{real}}$. The final code entry point (line 16) depicts one batching round, in which a node samples messages from pools, verifies the traffic rates constraints, and sends messages. Although, in an actual protocol implementation, a batching round would be triggered every $t_{\mathcal{P}}$ seconds, here, this is modeled by clock-ticks inputs. Indeed, in the UC framework, there is no notion of time, and the constraint "every $t_{\mathcal{P}}$ seconds" can not be formulated in a UC protocol or ideal functionality. Here, we assume that the clock-tick inputs are given out by the environment, on a regular basis (and passed down by the parent ITIs of Π_{link}, if the later is used as subroutine). The unfolding of a round follows the description given in Chapter 4 (p. 68), only transformed into pseudo-code: insert a dummy message in a fraction f_{dum} of pools, sample a batch of $n = \min_i(\min(n_{\mathcal{P}_{Y_i}} - n_{\mathcal{P}min}, n_{\mathcal{P}_{Y_i}} \cdot f_{\mathcal{P}}))$ messages from each pool, compute the traffic rates constraints (including *dummy budgets* $b_{I_{dum}}$ and $b_{O_{dum}}$), adjust the batches accordingly, update traffic counters, and send the resulting batches. The only difference with the description from Chapter 4 is that it is assumed that end-to-end dummies are not needed, *i.e.* that a node never receives *too many* messages over short periods of time (see comment on line 30). Indeed, this would otherwise necessitate Π_{link} to have access to routing tables (in order to find a suitable route on which to send the end-to-end dummy messages), in turn requiring to modify Π_{rtprop} and Π. Also, note that this assumption is realistic, since our results from practical experimentations on the protocol (Chapter 6, p. 140) show that nodes need only rarely to resort to end-to-end dummy messages.

B.7.2. Modeling a Variant of $\mathcal{F}_{\mathsf{link}}$

We then present a variant of $\mathcal{F}_{\mathsf{link}}$, denoted $\mathcal{F}'_{\mathsf{link}}$. Indeed, to be able to prove that Π_{link} UC-realises $\mathcal{F}_{\mathsf{link}}$, both ITIs should at least have the same input/output behavior. Yet, $\mathcal{F}_{\mathsf{link}}$, as described in Chapter 5, does not expect the same form of link-send input (Π_{link} requires a *ssid* and a c_{one} ciphertext to be provided in those inputs, while $\mathcal{F}_{\mathsf{link}}$ does not), and $\mathcal{F}_{\mathsf{link}}$ is not driven according to discrete *rounds* triggered by clock-tick inputs.

The adapted version of $\mathcal{F}_{\mathsf{link}}$ is given in Fig. B.4. Its differences with the version from Chapter 5 are the following. Firstly, $\mathcal{F}'_{\mathsf{link}}$ accepts a c_{one} ciphertext in link-send messages (although it does not use it). It no longer produces a link-rcvd message immediately after receiving a link-send input, but *emulates* delays. For corrupted nodes, this delay is specified by the ideal adversary $\mathcal{S}\texttt{im}$: $\mathcal{F}'_{\mathsf{link}}$ allows $\mathcal{S}\texttt{im}$ to specify when (at which round) the message should be output to Y, *via* a continue signal specifying the *ssid* of the link message. For honest nodes, $\mathcal{F}'_{\mathsf{link}}$ samples a delay $delay_r$ (in number of rounds), and stores the link-send input into a $pool_X$ structure, at index $r + delay_r$. This specifies that the message should be processed at round $r + delay_r$. Consequently, when rounds $r' = r + delay_r$ will be activated (that is, after $delay_r$ clock-tick inputs), $\mathcal{F}'_{\mathsf{link}}$ will look into $pool_X[r']$, and process all messages contained in this set. That is, for such message, $\mathcal{F}'_{\mathsf{link}}$ makes a link-rcvd output to the adequate neighbor of X. More generally, the functionality $\mathcal{F}'_{\mathsf{link}}$ checks the honest nodes' pools at every clock-tick input, and for every honest node.

The protocol is parameterised with $n_{\mathcal{P}min}$, $f_{\mathcal{P}}$, f_{dum}, Δr
upon input $(\mathsf{setup}, sid, neighbors)$:
  Initialise the round counter $r = 0$, and set $NT = neighbors$
  For each $Y_i \in NT$, create a pool $\mathcal{P}_{Y_i} \leftarrow \emptyset$
  $\forall r'$, set $n_{I_{dum}}(Y_i, r') = 0$, $n_{O_{dum}}(Y_i, r') = 0$, $n_{I_{real}}(r') = 0$, and $n_{O_{real}}(r') = 0$.
  // Perform DHKA
  $\forall Y_i \in NT$, sample $a_i \leftarrow\$ \mathbb{Z}_q$, store $(\mathsf{dhka}, a_i, Y_i)$, and send Y_i $(\mathsf{dhka}, sid, g^{a_i})$
upon receiving message $(\mathsf{dhka}, sid, g^b)$ from $Y \in NT$:
  Retrieve (dhka, a_i, Y), compute $s = (g^b)^{a_i}$, derives k_{XY}, k_{YX} from s, and store them.
upon input $(\mathsf{link\text{-}send}, sid, ssid, Y, m = \langle h, c, c' \rangle, c_{\mathsf{one}})$ (with a new $ssid$):
  Put $(sid, ssid, \left\langle \{h\}_{k_{XY}}, \mathsf{ReEnc}_{\mathsf{nopk}}(c_{\mathsf{one}}, c), \mathsf{ReEnc}_{\mathsf{nopk}}(c_{\mathsf{one}}, c') \right\rangle, \mathtt{type} \coloneqq \mathtt{real})$ in $\mathcal{P}_Y$.
upon receiving message $(\mathsf{link\text{-}msg}, sid, ssid, m)$ from Y (with a new $ssid$):
  Decrypt the header of m with stored key k_{YX} and obtain m'.
  if the header specifies a dummy flag **then** Increment $n_{I_{dum}}(Y, r)$
  else Increment $n_{I_{real}}(r)$, and output $(\mathsf{link\text{-}rcvd}, sid, ssid, Y, m')$.
upon input $(\mathsf{clock\text{-}tick}, sid)$: // Description of a batching round
  – Randomly sample a subset $\mathbf{P} \subseteq \left\{\mathcal{P}_{Y_i}\right\}_{\forall Y_i}$ of size $\lfloor |NT| \cdot f_{dum} \rfloor$.
    Generate $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
    In each $\mathcal{P}_{Y_i} \in \mathbf{P}$, insert $(sid, ssid_i, \left\langle \{\mathtt{dummy}\}_{k_{XY_i}}, \mathsf{Enc}(pk, 1), \mathsf{Enc}(pk, 1) \right\rangle, \mathtt{type} \coloneqq \mathtt{dummy})$
  – Let $k \coloneqq \min_i(\min(n_{\mathcal{P}_{Y_i}} - n_{\mathcal{P}min}, n_{\mathcal{P}_{Y_i}} \cdot f_{\mathcal{P}}))$
    if $k \neq 0$ **then** For each $\mathcal{P}_{Y_i}$, sample a set $M_{\mathcal{P}_{Y_i}}$ of k random messages.
    else $\forall Y_i \in NT$, $M_{\mathcal{P}_{Y_i}} = \left\{(sid, ssid_i, \left\langle \{\mathtt{dummy}\}_{k_{XY_i}}, \mathsf{Enc}(pk, 1), \mathsf{Enc}(pk, 1) \right\rangle, \mathtt{type} \coloneqq \mathtt{dummy})\right\}$
  – Compute:
    $equilibirum \coloneqq n_{O_{real}}(r - \Delta r) - n_{I_{real}}(r - \Delta r)$
    $b_{I_{dum}} \coloneqq \min_{Y_i \in NT}\left(\sum_{r' \in [r-\Delta r, r]} n_{I_{dum}}(Y_i, r')\right)$ and $b_{O_{dum}} \coloneqq \min_{Y_i \in NT}\left(\sum_{r' \in [r-\Delta r, r]} n_{O_{dum}}(Y_i, r')\right)$
    $max_{real} \coloneqq b_{I_{dum}} - equilibirum$ and $min_{real} \coloneqq -b_{O_{dum}} - equilibirum$
  – **while** $\sum_{Y_i \in NT}\left(\#\ \mathtt{real}\ \text{msgs in } M_{\mathcal{P}_{Y_i}}\right) > max_{real}$:
      Replace a random real message by a dummy in $M_{\mathcal{P}_{Y_i}}$, for a random Y_i
    while $\sum_{Y_i \in NT}\left(\#\ \mathtt{real}\ \text{msgs in } M_{\mathcal{P}_{Y_i}}\right) + \min_{Y_i \in NT}\left(\#\ \mathtt{dummy}\ \text{msgs in } M_{\mathcal{P}_{Y_i}}\right) < min_{real}$:
      // Assumed not to happen (would require end-to-end dummy message)
  – Decrease counters $n_{I_{dum}}$ and $n_{O_{dum}}$ according to the consumed *budgets*
    Set $n_{O_{real}}(r) \leftarrow \sum_{Y_i \in NT}\left(\#\ \mathtt{real} \in M_{\mathcal{P}_{Y_i}}\right)$ and $\forall Y_i \in NT$, $n_{O_{dum}}(Y_i, r) \leftarrow \left(\#\ \mathtt{dummy} \in M_{\mathcal{P}_{Y_i}}\right)$
  – $\forall M_{\mathcal{P}_{Y_i}}$ and $\forall (sid, ssid_j, m_j, \mathtt{type}_j) \in M_{\mathcal{P}_{Y_i}}$, send $(\mathsf{link\text{-}msg}, sid, ssid_j, m_j)$ to Y_i
  – Increment the round number r

Figure B.3. – Description of Π_{link} for node X

This design of the delays of honest nodes' messages, with the probability distribution χ, is discussed in the proof sketch below. It can be considered that χ corresponds to the discrete probability distribution of *pool delays* presented in Fig. 6.3 (p. 144), obtained empirically from simulations of our *proof-of-concept* protocol implementation.

Impact of $\mathcal{F}_{\mathsf{link}}$ on the Proofs in Chapter 5 Note that these modifications to $\mathcal{F}_{\mathsf{link}}$ require to slightly modify the proofs and proofs setup of Theorems 2 and 4 (that Π_{rtprop} and Π respectively UC-realise $\mathcal{F}_{\mathsf{rtprop}}$ and $\mathcal{F}$). In particular, the clock-tick inputs must be handled by all ITIs, and passed down by nodes' ITIs to $\mathcal{F}'_{\mathsf{link}}$. However, the proof outline stays the same, and does not modify the results. Indeed, the impacts on the

The functionality $\mathcal{F}_{\mathsf{link}}$ is responsible for delivering link messages.
upon input $(\mathsf{setup}, sid, neighbors)$ from party X:
  Initialise the round counter $r_X = 0$, and a message pool $pool_X$.
  Send $(\mathsf{dhka}, X, neighbors)$ to $\mathcal{S}\mathtt{im}$.
upon input $(\mathsf{link\text{-}send}, sid, ssid, Y, m)$ from party X (with a new $ssid$):
  if $X \in \Omega_c$ **then**
    Wait for $(\mathsf{continue}, sid, ssid)$ from $\mathcal{S}\mathtt{im}$.
    When received, output $(\mathsf{link\text{-}rcvd}, sid, ssid, X, m)$ to party Y in the current round.
  else
    Sample $delay_r \leftarrow\!\!\$\ \chi$, and store $(Y, (\mathsf{link\text{-}rcvd}, sid, ssid, X, m))$ in $pool_X[r + delay_r]$.
upon input $(\mathsf{clock\text{-}tick}, sid)$ from X:
  if $X \in \Omega_{\mathcal{S}\mathtt{im}}$ **then**
    For each $(Y, (\mathsf{link\text{-}rcvd}, sid, ssid, X, m))$ in $pool_X[r_X]$:
      output $(\mathsf{link\text{-}rcvd}, sid, ssid, X, m)$ to party Y.
    Send $(\mathsf{clock\text{-}tick}, sid, X)$ to $\mathcal{S}\mathtt{im}$.
    Increment the round counter r_X.

Figure B.4. – The Modified Link Message Functionality $\mathcal{F}'_{\mathsf{link}}$

real executions is only that link messages are not immediately transmitted, and may be out of order, and that $\mathcal{A}$ is allowed to specify the delay of its messages using $\mathsf{continue}$ messages to $\mathcal{F}'_{\mathsf{link}}$. This however do not necessitate a modification of Π_{rtprop} or Π. Then, the impact on the ideal execution (and in the construction of the simulator for the proof), is minor, since $\mathcal{S}\mathtt{im}$ simply needs to honestly simulate $\mathcal{F}'_{\mathsf{link}}$ instead of $\mathcal{F}_{\mathsf{link}}$, and let $\mathcal{A}$ specify the delays of corrupted node's messages. Apart from these modifications, the construction of the simulator does not need to change. Indeed, the simulator does not rely on the order in which messages are delivered, or on the fact that link messages are immediately delivered. Thus, the proofs of Theorems 2 and 4 are not impacted .

B.7.3. Towards showing that Π_{link} UC-realises $\mathcal{F}_{\mathsf{link}}$

Since it could not be proven, we do not formulate a theorem stating that Π_{link} UC-realises $\mathcal{F}'_{\mathsf{link}}$, but merely provide a proof sketch that it does.

Proof Sketch 7 (Π_{link} UC-realises $\mathcal{F}'_{\mathsf{link}}$). Similarly to previous proofs, the adversary $\mathcal{A}$ is assumed static and semi-honest, and we construct an adversary $\mathcal{S}\mathtt{im}$ that interacts with $\mathcal{E}$ and $\mathcal{F}'_{\mathsf{link}}$. $\mathcal{S}\mathtt{im}$ internally runs a copy of $\mathcal{A}$, and honestly forwards all communications between $\mathcal{A}$ and $\mathcal{E}$. $\mathcal{S}\mathtt{im}$ needs to simulate: the DHKAs, the real and dummy link messages sent and received by honest and corrupted nodes, and more importantly, $\mathcal{S}\mathtt{im}$ must simulate the rounds, and the fact that messages are delayed (to simulate the time they wait in pools in the real execution with Π_{link}). Note that, contrarily to the case of Π_{rtprop} and Π, here, actual messages are exchanged between the nodes' ITIs in a real execution of Π_{link}. This implies, in the UC framework, that $\mathcal{A}$ can observe these messages in the real execution (but not modify them, since we assume a passive adversary), and that these

messages must be simulated by $\mathcal{S}\mathtt{im}$ in the ideal execution. Therefore $\mathcal{S}\mathtt{im}$ must even simulate messages exchanged between honest nodes, including dummy ones.

$\mathcal{S}\mathtt{im}$ is constructed as follows. The DHKAs are the simplest component to simulate. When $\mathcal{S}\mathtt{im}$ externally receives $(\mathsf{dhka}, X, neighbors)$ from $\mathcal{F}'_{\mathsf{link}}$, in which X is a honest node, $\mathcal{S}\mathtt{im}(X)$ does the following for $Y \in neighbors$: it generates $a \leftarrow_{\$} \mathbb{Z}_q$, and simulates the sending of $(\mathsf{dhka}, sid, g^a)$ to Y. At some point, $\mathcal{S}\mathtt{im}(X)$ will receive a message $(\mathsf{dhka}, sid, g^b)$, from $\mathcal{A}(Y)$ if Y is corrupted, and played by $\mathcal{A}$, or from $\mathcal{S}\mathtt{im}(Y)$ if Y is honest and simulated by $\mathcal{S}\mathtt{im}$. Node $\mathcal{S}\mathtt{im}(X)$ then deduces $s = (g^a)^b$, and generates the link keys exactly as X would do in Π_{link}. In the rest of the simulation, $\mathcal{S}\mathtt{im}(X)$ encrypts the header of each link message between X and Y with these keys, and decrypts received messages' headers with them. Note that for DHKAs between two corrupted nodes, $\mathcal{S}\mathtt{im}$ has nothing to simulate, since $\mathcal{A}$ will play both neighboring nodes. At the end of the simulations of the DHKAs, note that $\mathcal{S}\mathtt{im}$ will have seen all neighbor values in setup *inputs, for both corrupted and honest nodes. This allows $\mathcal{S}\mathtt{im}$ to have knowledge of the whole topology graph.*

Then $\mathcal{S}\mathtt{im}$ simulates the exchanges of dummy and real link messages between nodes. We describe how this is done in the four possibles cases: messages between two corrupted nodes, messages from a corrupted node to a honest node, messages from a honest node to a corrupted node, and messages between honest nodes.

Simulating messages from $\mathcal{A}(X)$ to $\mathcal{A}(Y)$: *When the environment $\mathcal{E}$ provides a $(\mathsf{link\text{-}send}, sid, ssid, Y, m, c_{\text{one}})$ input to $\mathcal{A}(X)$, $\mathcal{S}\mathtt{im}$ immediately provides a copy of this input to $\mathcal{F}'_{\mathsf{link}}(X)$. $\mathcal{F}'_{\mathsf{link}}$ will then expect a $(\mathsf{continue}, sid, ssid)$ signal from $\mathcal{S}\mathtt{im}$. When $\mathcal{A}(Y)$ later makes the corresponding $(\mathsf{link\text{-}rcvd}, sid, ssid, X, m)$ outputs (which is linked to the* link-send *input, at least by the ssid value), $\mathcal{S}\mathtt{im}$ sends $(\mathsf{continue}, sid, ssid)$ to $\mathcal{F}'_{\mathsf{link}}$. Consequently, $\mathcal{F}'_{\mathsf{link}}(Y)$ outputs $(\mathsf{link\text{-}rcvd}, sid, ssid, X, m)$ to $\mathcal{S}\mathtt{im}$, which the latter simply discards.*

Note that this simulation was for the case of real *messages. However, $\mathcal{A}(X)$ also sends* dummy *messages to $\mathcal{A}(Y)$. Note that $\mathcal{A}(X)$ is not instructed by $\mathcal{E}$ to send dummy messages. Indeed, as per the code of Π_{rtprop} (that $\mathcal{A}(X)$ runs honestly since the adversary is assumed passive), $\mathcal{A}(X)$ crafts and sends such messages on its own. When $\mathcal{A}(X)$ sends a dummy message to $\mathcal{A}(Y)$, however, $\mathcal{S}\mathtt{im}$ has actually nothing to do: the exchange is internal to $\mathcal{A}$.*

This simulation of real and dummy messages from $\mathcal{A}(X)$ to $\mathcal{A}(Y)$ perfectly mirrors a real execution, and is thus indistinguishable from it.

Simulating messages from $\mathcal{A}(X)$ to $\mathcal{S}\mathtt{im}(Y)$: *Here, $\mathcal{S}\mathtt{im}$ must play the role of the link-receiver Y. Again, we distinguish the case of real messages and the case of dummy messages. The simulation of real messages is analogous to the above simulation case, except that, at the end, $\mathcal{F}'_{\mathsf{link}}(Y)$ makes the* link-rcvd *output directly to $\mathcal{E}$. The fact that $\mathcal{S}\mathtt{im}$ can specify by a* continue *message is crucial here: it allows to ensure that $\mathcal{F}_{\mathsf{link}}(Y)$ makes its* link-rcvd *only after $\mathcal{A}(X)$ actually sent the message.*

For dummy messages, the simulation is trivial: when $\mathcal{S}\mathtt{im}(Y)$ receives a dummy message from $\mathcal{A}(X)$, $\mathcal{S}\mathtt{im}$ simply discards it.

Again, these simulations perfectly mirror a real execution.

Simulating messages from $\mathcal{S}\mathtt{im}(X)$ ***to*** $\mathcal{A}(Y)$***:*** *From this point on, we have to introduce strong assumptions to carry out the proofs. Indeed, as we will see, it is not possible to construct a simulator and prove indistinguishability with a real execution without such assumptions.*

For the case of messages from a honest nodes to a corrupted neighbor, we again distinguish between the sending of real and the sending of dummy messages, and begin with the case of real messages. $\mathcal{S}\mathtt{im}$ *does not get to see the* link-send *input for honest nodes. However, when the link-receiver* Y *is corrupted,* $\mathcal{S}\mathtt{im}$ *receives from* $\mathcal{F}'_{\mathsf{link}}(Y)$ *the output* $(\mathsf{link\text{-}rcvd}, sid, ssid, X, m = \langle h, c_1, c_2 \rangle)$*, with a header* h *in the clear (not AES encrypted with link keys). Consequently,* $\mathcal{S}\mathtt{im}$ *knows that it must simulate the sending of a real message to* $\mathcal{A}(Y)$*. For that,* $\mathcal{S}\mathtt{im}(X)$ *crafts* $m' = \left\langle \{h\}_{k_{XY}}, c_1, c_2 \right\rangle$ *(encrypting* h *with the link key it knows, and copying* c_1 *and* c_2 *from the* link-rcvd *outputs).* $\mathcal{S}\mathtt{im}(X)$ *then sends* $(\mathsf{link\text{-}msg}, sid, ssid, m')$ *to* $\mathcal{A}(Y)$*. This simulation of real messages from* $\mathcal{S}\mathtt{im}(X)$ *to* $\mathcal{A}(Y)$ *is however not indistinguishable from a real execution. Indeed,* $\mathcal{F}'_{\mathsf{link}}$ *is constructed to* emulate *the delays of the pool-based message re-ordering in* Π_{link} *by sampling a delay from the probability distribution* χ*. However, even if* χ *can be approximated by experimental measures, it is quite likely that there will be a non-negligible statistical distance between the messages delays in the real execution and the delays emulated by* $\mathcal{F}'_{\mathsf{link}}$*. Indeed,* $\mathcal{F}'_{\mathsf{link}}$ *does not take into account, in particular, the number of received and sent real messages in past rounds, and thus may not even respect the traffic rates constraints. To be able to carry on with the proof, we will however assume, for the sake of the argument, that the independently sampling the delay ofr each real message from* χ *produces delays that are indistinguishable from that of a real execution. This assumption is discussed after the proof sketch.*

Now, for the case of dummy messages from a honest node to a corrupted one, the simulation is also inevitably flawed. Indeed, $\mathcal{S}\mathtt{im}$ *does not get any* hint *that it must simulate the sending of a dummy message. Instead,* $\mathcal{S}\mathtt{im}$ *must, by itself, know how many dummy messages* $\mathcal{S}\mathtt{im}(X)$ *should send to* $\mathcal{A}(Y)$ *in each round. However, this number varies from round to round, and in a real execution, it would depend on the values of the traffic counters* $n_{I_{real}}$*,* $n_{I_{dum}}$*,* $n_{O_{real}}$*,* $n_{O_{dum}}$ *of node* X*. The issue here, is that* $\mathcal{S}\mathtt{im}$ *does not know the value of these counters. Actually, simulating dummy messages from honest nodes is the second challenge of this proofs, and the second point that we do not know how to prove (and that may not be provable). For this reason, we defer its description in the next paragraph, after discussing the simulation of messages between two honest nodes.*

Simulating messages from $\mathcal{S}\mathtt{im}(X)$ ***to*** $\mathcal{S}\mathtt{im}(Y)$***:*** *The construction the simulator for this case involves the same difficulties as mentioned in the previous one. That is,* $\mathcal{S}\mathtt{im}$ *has no source of information regarding the messages exchanged between honest nodes. It does not know the number of messages that honest nodes exchange per round, nor the type (dummy or real) of these messages. In consequence,* $\mathcal{S}\mathtt{im}$

can not simulate link messages between honest nodes such that the simulation is indistinguishable from a real execution. As a side note, we remark that this is actually the desired property of our protocol, modeled as an assumption by $\mathcal{F}'_{\mathsf{link}}$: that an external adversary is not able to know the number of real messages exchanged between neighboring nodes.
We describe below how link messages from honest to honest nodes are simulated, in the same time as the simulation of dummy messages from honest to corrupted nodes.

The approach we suggest here is to approximate *the number of link messages sent by a honest node in each round. We do not distinguish real and dummy messages, but only care about the number of messages. More formally, we make the following assumption: the number of messages sent by a honest node at round r is dictated by a known probability distribution. That is, $\mathcal{S}\mathtt{im}$ knows, for any node X, a (possibly different) probability distribution $\psi_X : \mathbb{N} \to [0, 1]$, which associates a probability to any number of link messages. Therefore, we construct the simulation of messages sent by a honest node X in the following way. When $\mathcal{S}\mathtt{im}$ receives* $(\mathsf{clock\text{-}tick}, sid, X)$ *from $\mathcal{F}'_{\mathsf{link}}$, $\mathcal{S}\mathtt{im}$ samples a number of link messages n from ψ_X. For each neighbor Y of X, $\mathcal{S}\mathtt{im}$ does the following:*

- *If Y is honest, then $\mathcal{S}\mathtt{im}(X)$ crafts and sends to $\mathcal{S}\mathtt{im}(Y)$ n different messages $\left\langle \{\mathtt{dummy}\}_{k_{XY}}, \mathsf{Enc}(pk, 1), \mathsf{Enc}(pk, 1) \right\rangle$, where k_{XY} is the actual key for this pair of the X-Y link, and pk is a freshly generated public key. That is, $\mathcal{S}\mathtt{im}(X)$ sends only dummy messages.*

  Simulating messages between neighboring honest nodes in this manner ensures indistinguishability from a real execution, if we do not take into account the fact that the number *n of link message sent may not be* correct*. We discuss this latter point later, and show here that the* appearance *and data contained in messages do not allow to distinguish the real and simulated executions. Indeed, the fact that X always sends dummy messages is not detectable (except with a negligible probability). Although X may be instructed by $\mathcal{E}$ to send real messages, the fact that the encrypted header always contains a* $\mathtt{dummy}$ *flag (instead of an actual header such as* $\mathtt{payload}\|cid$*), and the fact that Elgamal ciphertexts encrypt* 1 *under a random public key, are undetectable for the adversary. Indeed, if there exist adversary distinguishing between the simulation and a real execution based on the encrypted header, then it is trivial to show that there exists an adversary breaking the IND-CPA security of the AES. Then,if there exist adversary distinguishing between the simulation and a real execution based on the Elgamal ciphertexts, then it is possible to construct an adversary breaking the USS property of the scheme. Indeed, although it is $\mathcal{E}$ that provides the ciphertexts c_1 and c_2 of real messages in* $\mathsf{link\text{-}send}$ *inputs to $\mathcal{F}'_{\mathsf{link}}(X)$, $\mathcal{E}$ does not expect to* recognise *them when they are sent in a* $\mathsf{link\text{-}msg}$ *message, since in the code of Π_{link}, a node re-encrypts every real message it receives as input.*

- *If Y is corrupted, then $\mathcal{S}\mathtt{im}(X)$ behaves exactly in the same way, but for a number of messages $n' \coloneqq n - n_{real}$, where n_{real} is the number of real link messages already*

simulated from $\mathcal{S}\mathtt{im}(X)$ to $\mathcal{A}(Y)$ in the current round. Indeed, as described in the simulation case "simulating messages from $\mathcal{S}\mathtt{im}(X)$ to $\mathcal{A}(Y)$", $\mathcal{S}\mathtt{im}$ gets a leak from $\mathcal{F}'_{\mathsf{link}}$ every time $\mathcal{S}\mathtt{im}(X)$ sends a real message to $\mathcal{A}(Y)$. We define n_{real} as the number of such leaks.

In both cases (whether Y is honest or corrupted), the simulation of link messages, in particular dummy ones, from a honest node X is not indistinguishable from a real execution. There are several things that can go wrong. *The most straightforward example is that, for the simulation of dummy messages from a honest node to a corrupted one, the sampled number of link message n may actually be smaller than the number n_{real} of already simulated real messages. More generally, proving that the simulation is indistinguishable from a real execution would require to prove that, at each round, the probability that the sampled number n of messages sent by honest node X does not correspond the number that this node would send in a real execution, is negligible. We do not know of a way to prove this fact, which may actually not be provable at all. Therefore, to conclude the proof (sketch), we put it as assumption. That is, we assume that, for each honest node X and each round r, the number n sampled from ψ_X is exactly equal to the number of link messages that X would send to each neighbor, except with a negligible probability.*

This concludes the proof sketch.

B.7.4. Perspectives

As a result, in order to complete the proof, we made two strong assumptions:

- That independently sampling the delay of each message using χ produces message delays indistinguishable from that of a real execution, except with negligible probability.
- That there exists a probability distribution ψ_X for each node X that gives the number of link messages (real and dummy counter together) that node X sends to each neighbor in a given round.

These two assumptions are likely to be impossible to prove. This means ultimately that it is not possible that Π_{link} UC-realises $\mathcal{F}'_{\mathsf{link}}$ (at least with the chosen approach for the proof). Intuitively, this means that $\mathcal{F}'_{\mathsf{link}}$ (and thus $\mathcal{F}_{\mathsf{link}}$) is also a strong assumption in itself. Although we intuitively knew that $\mathcal{F}_{\mathsf{link}}$ was indeed as strong assumption, we employed this ideal functionality in Chapter 5, and provided proofs in the $\mathcal{F}_{\mathsf{link}}$-hybrid model, so as to focus the analysis on the impact of (collusions of) corrupted nodes, that we believe are the greater threat to our protocol.

Our our attempt at proving that Π_{link} UC-realises $\mathcal{F}'_{\mathsf{link}}$ put in light the difficulty of formalising a security definition (*i.e.* an ideal functionality) that captures the fact that an external observer can not know the number of messages exchanged between neighboring nodes, and the difficulty of designing a protocol that UC-realises it.

This attempt also shows how $\mathcal{F}_{\mathsf{link}}$ should be modified to become UC-realisable; *i.e.* what are the additional leaks that $\mathcal{F}_{\mathsf{link}}$ must provide to $\mathcal{S}\mathtt{im}$ to be able to construct an adequate

simulator. In our approach here, we have proposed a variant $\mathcal{F}'_{\mathsf{link}}$ that conserves the results from Chapter 5. That is, $\mathcal{F}'_{\mathsf{link}}$ is a variant of the ideal functionality used in Chapter 5, but that does not invalidate our proofs of Theorems 2 and 4. However, if we accept to break this compatibility, here are some leads to modify the functionality so as to make it UC-realisable:

- Firstly, run Π_{link} inside $\mathcal{F}_{\mathsf{link}}$, so that the latter can perfectly emulate the delays of honest nodes' real messages, instead of relying on χ.
- Since, in a real-world run of our protocol over the Internet, a network observer does see the number of link messages sent by a node, it would be logical to make $\mathcal{F}_{\mathsf{link}}$ leak that information (without specifying how many of these messages are real ones, and how many are dummy ones). This can be done also by having $\mathcal{F}_{\mathsf{link}}$ internally run an instance of Π_{link}, and observe messages sent by nodes at each round.
- These two previous modifications, however, require that the simulated execution and the internal state of $\mathcal{F}_{\mathsf{link}}$ be *synchronised*. In particular, it is necessary that $\mathcal{F}_{\mathsf{link}}$ knows the number of dummy messages sent by corrupted nodes to honest neighbors. Otherwise, the internal Π_{link} instance can not run, since honest nodes with corrupted neighbors can not properly compute their dummy budget $b_{I_{dum}}$. Yet, the number of dummy messages sent by corrupted nodes is determined by $\mathcal{A}$ (ran within $\mathcal{S}\mathtt{im}$ in the ideal execution). Although $\mathcal{A}$ is a passive adversary and follows the code of Π_{link}, it remains that the number of dummy messages that a corrupted node sends depends on $\mathcal{A}$'s random choices, and that these choices must be communicated to $\mathcal{F}_{\mathsf{link}}$. Thus, $\mathcal{F}_{\mathsf{link}}$ should be modified to let $\mathcal{S}\mathtt{im}$ specify the number of dummy messages that each corrupted node $\mathcal{A}(X)$ sends in each round.

To further validate our proofs of security, we envision to design a new variant of $\mathcal{F}_{\mathsf{link}}$ following these guidelines, verify that Π_{link} UC-realises it, and then adapt the proofs of Theorems 2 and 4 to the $\mathcal{F}_{\mathsf{link}}$-hybrid model for this new version.

Publications

[GBP13a] Antoine Guellier, Christophe Bidan, and Nicolas Prigent. “Protocole Ad Hoc Proactif Anonyme à Base de Cryptographie Homomorphique”. In: *SAR-SSI*. Mont de Marsan, France, September 2013.

[GBP13b] Antoine Guellier, Christophe Bidan, and Nicolas Prigent. “Routage Ad Hoc Proactif Respectueux de La Vie Privée”. In: *Atelier Sur La Protection de La Vie Privée*. 2013.

[Gue14] Antoine Guellier. *Can Homomorphic Cryptography Ensure Privacy?* RR-8568. Inria, July 2014, page 111. URL: http://hal.inria.fr/hal-01052509.

[GBP14] Antoine Guellier, Christophe Bidan, and Nicolas Prigent. “Homomorphic Cryptography-Based Privacy-Preserving Network Communications”. In: *ATIS*. Edited by Lynn Batten. Volume 490. CCIS. Melbourne, Australia: Springer Berlin, Heidelberg, November 2014, pages 159–170.

[Alp+16] Gergely Alpár, Lejla Batina, Lynn Batten, Veelasha Moonsamy, Anna Krasnova, Antoine Guellier, and Iynkaran Natgunanathan. “New Directions in IoT Privacy Using Attribute-Based Authentication (Position Paper)”. In: *ACM Malicious Software and Hardware in Internet of Things Workshop*. Como, Italy: ACM Press, May 2016.

Bibliography

[AS16] Sebastian Angel and Srinath Setty. "Unobservable Communication over Fully Untrusted Infrastructure". In: *USENIX Conference on Operating Systems Design and Implementation*. Savannah, GA, USA: USENIX Association, November 2016, pages 551–569 (cited on pages 13, 28, 33, 83).

[AG12] Mohd Anwar and Jim Greer. "Role- and Relationship-Based Identity Management for Privacy-Enhanced E-Learning". In: *International Journal of Artificial Intelligence in Education* 21.3 (January 1, 2012), pages 191–213 (cited on page 53).

[Bac+13] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi. "AnoA: A Framework for Analyzing Anonymous Communication Protocols". In: *Computer Security Foundations Symposium*. New Orleans, LA, USA: IEEE Computer Society, June 2013, pages 163–178 (cited on pages 92, 95, 97, 128, 130, 131, 183).

[Bac+12] Michael Backes, Ian Goldberg, Aniket Kate, and Esfandiar Mohammadi. "Provably Secure and Practical Onion Routing". In: *Computer Security Foundations*. Cambridge, MA, USA: IEEE Computer Society, June 2012, pages 369–385 (cited on pages 92, 94, 96, 107).

[BMS16] Michael Backes, Sebastian Meiser, and Marcin Slowik. "Your Choice MATor(S)". In: *Proceedings on Privacy Enhancing Technologies* 2016.2 (April 2016), pages 40–60 (cited on pages 43, 92, 94, 101, 135, 147).

[Bel+01] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. "Key-Privacy in Public-Key Encryption". In: *ASIACRYPT*. Edited by Colin Boyd. Volume 2248. LNCS. Gold Coast, Australia: Springer Berlin, Heidelberg, December 2001, pages 566–582 (cited on pages 22, 102).

[BG03] Krista Bennett and Christian Grothoff. "GAP - Practical Anonymous Networking". In: *Privacy Enhancing Technologies*. Edited by Roger Dingledine. Volume 2760. LNCS. Dresden, Germany: Springer Berlin Heidelberg, March 2003, pages 141–160 (cited on pages 28, 37).

[BPS01] Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. "The Disadvantages of Free MIX Routes and How to Overcome Them". In: *Designing Privacy Enhancing Technologies*. Edited by Hannes Federrath. LNCS 2009. Berkeley, CA, USA: Springer Berlin Heidelberg, July 2001, pages 30–45 (cited on pages 41, 68, 151).

[Ber+11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles van Assche. *The Keccak Reference*. NIST Submission (3.0). January 14, 2011, page 69 (cited on pages 20, 103).

[Boc+12] Simon Boche, Christophe Bidan, Sébastien Gambs, and Nicolas Prigent. "Protection de La Vie Privée Dans Les Réseaux Mobiles Ubiquitaires". Master Thesis. Université de Rennes 1, Supélec, June 21, 2012 (cited on page 81).

[Boh+04] Rainer Böhme, George Danezis, Claudia Diaz, Stefan Köpsell, and Andreas Pfitzmann. *On the PET Workshop Panel "Mix Cascades versus Peer-to-Peer: Is One Concept Superior?"* 2004 (cited on page 45).

[Bon98] Dan Boneh. "The Decision Diffie-Hellman Problem". In: *Algorithmic Number Theory*. Edited by Joe P. Buhler. Volume 1423. LNCS. Portland, Oregon, USA: Springer Berlin Heidelberg, June 21, 1998, pages 48–63 (cited on pages 18, 155, 156).

[CL05] Jan Camenisch and Anna Lysyanskaya. "A Formal Treatment of Onion Routing". In: *CRYPTO*. Edited by Victor Shoup. Volume 3621. LNCS. Santa Barbara, California, USA: Springer Berlin Heidelberg, August 2005, pages 169–187 (cited on page 96).

[Can13] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. Cryptology ePrint Archive: Report 2000/067. 2013 (cited on pages 92, 93, 159, 160).

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. "A Simpler Variant of Universally Composable Security for Standard Multiparty Computation". In: *CRYPTO*. Edited by Rosario Gennaro and Matthew Robshaw. Volume 9216. LNCS. Santa Barbara, CA, USA: Springer Berlin Heidelberg, August 16, 2015, pages 3–22 (cited on page 93).

[Can+02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. *Universally Composable Two-Party and Multi-Party Secure Computation*. Cryptology ePrint Archive: Report 2002/140. 2002 (cited on pages 112, 159, 160).

[CSV16] Ran Canetti, Daniel Shahaf, and Margarita Vald. "Universally Composable Authentication and Key-Exchange with Global PKI". In: *Public-Key Cryptography*. Edited by Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang. Volume 9615. LNCS. Taipei, Taiwan: Springer Berlin Heidelberg, March 6, 2016, pages 265–296 (cited on page 93).

[Cha81] David L. Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Communication of the ACM* 24.2 (February 1981), pages 84–90 (cited on pages 2, 8, 29, 32, 36, 37).

[Cha+16] David Chaum, Farid Javani, Aniket Kate, Anna Krasnova, Joeri de Ruiter, and Alan T. Sherman. *cMix: Anonymization by High-Performance Scalable Mixing*. Cryptology ePrint Archive: Report 2016/008. 2016 (cited on pages 27, 28, 32, 34, 92, 94, 95, 107, 134).

[Che+15] Chen Chen, Daniele E. Asoni, David Barrera, George Danezis, and Adrain Perrig. "HORNET: High-Speed Onion Routing at the Network Layer". In: *Computer and Communications Security*. Denver, Colorado, USA: ACM Press, October 2015, pages 1441–1454 (cited on pages 14, 28).

[CPP06] Benoît Chevallier-Mames, Pascal Paillier, and David Pointcheval. "Encoding-Free ElGamal Encryption Without Random Oracles". In: *Public Key Cryptography*. Edited by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. Volume 3958. LNCS. New York, NY, USA: Springer Berlin, Heidelberg, April 2006, pages 91–104 (cited on pages 155, 156).

[Cla+10] Ian Clarke, Oskar Sandberg, Matthew Toseland, and Vilhelm Verendel. *Private Communication Through a Network of Trusted Connections: The Dark Freenet*. Unpublished. 2010 (cited on pages 10, 57, 152).

[Cla+01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. "Freenet: A Distributed Anonymous Information Storage and Retrieval System". In: *Designing Privacy Enhancing Technologies*. Edited by Hannes Federrath. Volume 2009. LNCS. Springer Berlin, Heidelberg, 2001, pages 46–66 (cited on page 27).

[CKK05] Sebastian Clauβ, Dogan Kesdogan, and Tobias Kölsch. "Privacy Enhancing Identity Management: Protection Against Re-Identification and Profiling". In: *Workshop on Digital Identity Management*. Fairfax, VA, USA: ACM Press, 2005, pages 84–93 (cited on page 53).

[CBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazieres. "Riposte: An Anonymous Messaging System Handling Millions of Users". In: *Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 2015, pages 321–338 (cited on page 33).

[DC06] G. Danezis and R. Clayton. "Route Fingerprinting in Anonymous Communications". In: *Peer-to-Peer Computing*. Cambridge, UK: IEEE Computer Society, September 2006, pages 69–72 (cited on page 43).

[Dan03a] George Danezis. "Mix-Networks with Restricted Routes". In: *Privacy Enhancing Technologies*. Edited by Roger Dingledine. Volume 2760. LNCS. Dresden, Germany: Springer Berlin Heidelberg, March 26, 2003, pages 1–17 (cited on pages 52, 57).

[Dan03b] George Danezis. "Statistical Disclosure Attacks: Traffic Confirmation in Open Environments". In: *Security and Privacy in the Age of Uncertainty*. Edited by Dimitris Gritzalis, Sabrina De Capitani di Vimercati, Pierangela Samarati, and Sokratis Katsikas. IFIP International Federation for Information Processing 122. Springer US, 2003, pages 421–426 (cited on page 44).

[Dan04] George Danezis. "The Traffic Analysis of Continuous-Time Mixes". In: *Privacy Enhancing Technologies*. Edited by David Martin and Andrei Serjantov. Volume 3424. LNCS. Toronto, Canada: Springer Berlin Heidelberg, May 2004, pages 35–50 (cited on pages 33, 44, 46).

[Dan06] George Danezis. "Breaking Four Mix-Related Schemes Based on Universal Re-Encryption". In: *Information Security*. LNCS. Samos Island, Greece: Springer Berlin Heidelberg, September 2006, pages 46–59 (cited on page 25).

[DDM03] George Danezis, Roger Dingledine, and Nick Mathewson. "Mixminion: Design of a Type III Anonymous Remailer Protocol". In: *Security and Privacy*. Berkeley, CA, USA: IEEE Computer Society, May 2003, pages 2–15 (cited on pages 28, 29, 31, 32, 38, 43).

[DS08] George Danezis and Paul Syverson. "Bridging and Fingerprinting: Epistemic Attacks on Route Selection". In: *Privacy Enhancing Technologies*. Edited by Nikita Borisov and Ian Goldberg. Volume 5134. LNCS. Springer Berlin Heidelberg, July 23, 2008, pages 151–166 (cited on page 43).

[DG09] Georges Danezis and Ian Goldberg. "Sphinx: A Compact and Provably Secure Mix Format". In: *Security and Privacy*. Berkely, CA, USA: IEEE Computer Society, May 2009, pages 269–282 (cited on pages 29, 32, 96).

[DP04] Claudia Diaz and Bart Preneel. "Taxonomy of Mixes and Dummy Traffic". In: *Information Security Management, Education and Privacy*. Edited by Yves Deswarte, Frédéric Cuppens, Sushil Jajodia, and Lingyu Wang. Volume 148. IFIP International Federation for Information Processing. Springer US, 2004, pages 217–232 (cited on pages 3, 34, 49, 62).

[DH76] W. Diffie and M. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (November 1976), pages 644–654 (cited on page 20).

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". In: *USENIX Security Symposium*. Volume 13. San Diego, CA: USENIX Association, August 2004, pages 21–21 (cited on pages 2, 27–31, 43, 49, 71, 83, 135, 151).

[DSS04] Roger Dingledine, Vitaly Shmatikov, and Paul Syverson. "Synchronous Batching: From Cascades to Free Routes". In: *Privacy Enhancing Technologies*. Edited by David Martin and Andrei Serjantov. Volume 3424. LNCS. Toronto, Canada: Springer Berlin Heidelberg, May 26, 2004, pages 186–206 (cited on pages 34, 44).

[Daz+03] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. "Towards Measuring Anonymity". In: *Privacy Enhancing Technologies*. Edited by Roger Dingledine and Paul Syverson. Volume 2482. LNCS. Springer Berlin, Heidelberg, 2003, pages 54–68 (cited on page 145).

[Don+09] Ying Dong, Tat Wing Chim, Victor O. K. Li, S. M. Yiu, and C. K. Hui. "ARMR: Anonymous Routing Protocol with Multiple Routes for Communications in Mobile Ad Hoc Networks". In: *Privacy and Security in Wireless Sensor and Ad Hoc Networks* 7.8 (November 2009), pages 1536–1550 (cited on page 81).

[Elg85] Taher Elgamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *CRYPTO*. Edited by George Robert Blakley and David Chaum. Volume 196. LNCS. Santa Barbara, CA, USA: Springer Berlin, Heidelberg, August 1985, pages 10–18 (cited on page 22).

[Eur95] European Parliament and Council of the Europen Union. *Directive 1995/46/EC*. On the protection of individuals with regard to the processing of personal data and on the free movement of such data. October 1995 (cited on page 1).

[Eur02] European Parliament and Council of the Europen Union. *Directive 2002/58/EC*. Concerning the processing of personal data and the protection of privacy in the electronic communications sector. July 2002 (cited on page 1).

[Eur16] European Parliament and Council of the Europen Union. *Directive 2016/679*. On the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC. April 2016 (cited on page 1).

[EG11] Nathan S. Evans and Christian Grothoff. "R5N: Randomized Recursive Routing for Restricted-Route Networks". In: *5th International Conference on Network and System Security*. Milan, Italy: IEEE Computer Society, September 2011, pages 316–321 (cited on page 10).

[Fis+10] Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. "Random Oracles with(out) Programmability". In: *ASIACRYPT*. Edited by Masayuki Abe. Volume 6477. LNCS. Singapore: Springer Berlin Heidelberg, December 5, 2010, pages 303–320 (cited on page 103).

[Fre17] Free Haven. *Selected Papers in Anonymity*. Web Page. 2017. URL: https://www.freehaven.net/anonbib/ (visited on 03/10/2017) (cited on page 2).

[FM02] Michael J. Freedman and Robert Morris. "Tarzan: A Peer-to-Peer Anonymizing Network Layer". In: *Computer and Communications Security*. Washington, DC, USA: ACM Press, 2002, pages 193–206 (cited on pages 3, 10, 13, 38, 40, 51, 70, 71).

[Fre15] French Parliament. *Loi N° 2015-912 Du 24 Juillet 2015 Relative Au Renseignement*. Journal Officiel n° 0171 du 26 Juillet 2015. 2015 (cited on page 1).

[Gen09] Craig Gentry. "A Fully Homomorphic Encryption Scheme". Stanford, CA, USA: Stanford University, September 2009 (cited on page 21).

[Gha16] Taher Ahmed Ghaleb. "Techniques and Countermeasures of Website/Wireless Traffic Analysis and Fingerprinting". In: *Cluster Computing* 19.1 (March 2016), pages 427–438 (cited on pages 47, 48, 151).

[Gir15] Damien Giry. *Keylength - Cryptographic Key Length Recommendation*. Web Page. September 17, 2015. URL: https://www.keylength.com (visited on 07/07/2016) (cited on pages 18, 155).

[Glo16] Global Freedom of Expression, Columbia University. *The Case of Luxleaks*. June 29, 2016. URL: https://globalfreedomofexpression.columbia.edu/cases/the-case-of-luxleaks/ (visited on 06/06/2017) (cited on page 2).

[Gol01] Oded Goldreich. *The Foundations of Cryptography*. Volume 1 - Basic Tools. Cambridge University Press, June 2001. ISBN: 0-521-83084-2 (cited on pages 12, 92).

[Gol04] Oded Goldreich. *The Foundations of Cryptography*. Volume 2 - Basic Applications. Cambridge, MA, USA: Cambridge University Press, May 2004. ISBN: 0-521-83084-2 (cited on pages 19, 21, 152).

[Gol+04] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. "Universal Re-Encryption for Mixnets". In: *Topics in Cryptology–CT-RSA*. Edited by Tatsuaki Okamoto. Volume 2964. LNCS. San Francisco, CA, USA: Springer Berlin Heidelberg, February 2004, pages 163–178 (cited on pages 24, 25, 29, 103).

[GKK04] Gomułkiewicz, Marcin, Klonowski, Marek, and Kutyłowski, Mirosław. "Onions Based on Universal Re-Encryption–anonymous Communication Immune against Repetitive Attack". In: *Information Security Applications*. Edited by Chae Hoon Lim and Moti Yung. LNCS. Jeju Island, Korea: Springer Berlin Heidelberg, August 2004, pages 400–410 (cited on page 25).

[Gre14] Glenn Greenwald. *No Place to Hide*. Metropolitan Books, May 2014. 272 pages. ISBN: 978-1-62779-073-4 (cited on pages 1, 2, 12, 154).

[Gue17] Antoine Guellier. *APART: Proof-of-Concept Implementation of a Privacy-Preserving Internet Overlay*. GitHub Repository. 2017. URL: https://github.com/aguellier/apart (cited on page 138).

[GT96] Ceki Gülcü and Gene Tsudik. "Mixing E-Mail with Babel". In: *Network and Distributed System Security*. San Diego, CA, USA: IEEE Computer Society, February 1996, pages 2–16 (cited on pages 28, 32).

[HM08] Alejandro Hevia and Daniele Micciancio. "An Indistinguishability-Based Characterization of Anonymous Channels". In: *Privacy Enhancing Technologies*. Edited by Nikita Borisov and Ian Goldberg. Volume 5134. LNCS. Springer Berlin, Heidelberg, 2008, pages 24–43 (cited on page 95).

[Hoo+15] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. "Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis". In: *Symposium on Operating Systems Principles*. Monterey, CA, USA: ACM Press, 2015, pages 137–152 (cited on page 32).

[HB13] Amir Houmansadr and Nikita Borisov. "The Need for Flow Fingerprints to Link Correlated Network Flows". In: *Privacy Enhancing Technologies*. Edited by Emiliano De Cristofaro and Matthew Wright. Volume 7981. LNCS. Bloomington, IN, USA: Springer Berlin Heidelberg, July 10, 2013, pages 205–224 (cited on page 151).

[HBS13] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. "The Parrot Is Dead: Observing Unobservable Network Communications". In: *Security and Privacy*. Berkeley, CA, USA: IEEE Computer Society, May 2013, pages 65–79 (cited on page 152).

[HLF12] Cheng-Qiang Huang, Long-Hai Li, and Shao-Feng Fu. "An Improved Construction for Reusable Anonymous Return Channels Based on Universal Re-Encryption". In: *Cyber-Enabled Distributed Computing and Knowledge Discovery*. Sanya, China: IEEE Computer Society, October 2012, pages 155–159 (cited on pages 25, 42, 62).

[I2P] I2P. *(The Invisible Internet Project)*. URL: https://geti2p.net/en/docs/how/tech-intro (visited on 12/01/2015) (cited on pages 28, 151).

[IKV13] Luca Invernizzi, Christopher Kruegel, and Giovanni Vigna. "Message in a Bottle: Sailing Past Censorship". In: *Anual Computer Security Applications Conference*. New Orleans, Louisiana: ACM Press, 2013, pages 39–48 (cited on pages 83, 152).

[Jak99] Markus Jakobsson. "Flash Mixing". In: *ACM Symposium on Principles of Distributed Computing*. Atlanta, Georgia, USA: ACM Press, 1999, pages 83–89 (cited on page 24).

[Joy16] Marc Joye. "Secure ElGamal-Type Cryptosystems Without Message Encoding". In: *The New Codebreakers*. Edited by Peter Y. A. Ryan, David Naccache, and Jean-Jacques Quisquater. LNCS 9100. Springer Berlin Heidelberg, 2016, pages 470–478 (cited on page 156).

[Jua+15] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. *WTF-PAD: Toward an Efficient Website Fingerprinting Defense*. December 1, 2015. arXiv: 1512.00524 (cited on pages 47, 48).

[KAP02] Dogan Kedogan, Dakshi Agrawal, and Stefan Penz. "Limits of Anonymity in Open Environments". In: *Information Hiding*. Edited by Fabien A. P. Petitcolas. Volume 2578. LNCS. Noordwijkerhout, The Netherlands: Springer, October 2002, pages 53–69 (cited on page 44).

[Kes+06] Dogan Kesdogan, Dakshi Agrawal, Vinh Pham, and Dieter Rautenbach. "Fundamental Limits on the Anonymity Provided by the MIX Technique". In: *Security and Privacy*. Berkeley/Oakland, CA, USA: IEEE Computer Society, May 2006, pages (cited on pages 44, 45).

[KS05] Lea Kissner and Dawn Song. "Privacy-Preserving Set Operations". In: *CRYPTO*. Edited by Victor Shoup. Volume 3621. LNCS. Santa Barbara, CA, USA: Springer Berlin Heidelberg, August 14, 2005, pages 241–257 (cited on page 81).

[Kwo+15] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. "Riffle". In: *Proceedings on Privacy Enhancing Technologies* 2016.2 (2015), pages 115–134 (cited on page 32).

[Lee14] Micah Lee. *Ed Snowden Taught Me To Smuggle Secrets Past Incredible Danger. Now I Teach You.* Blog Post. October 28, 2014. URL: https://theintercept.com/2014/10/28/smuggling-snowden-secrets/ (visited on 07/20/2016) (cited on pages 151, 154).

[Lev+04] Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew Wright. "Timing Attacks in Low-Latency Mix Systems". In: *Financial Cryptography*. Edited by Ari Juels. Key West, FL, USA: Springer Berlin Heidelberg, February 2004, pages 251–265 (cited on page 46).

[Lin16a] Yehuda Lindell. *How To Simulate It - A Tutorial on the Simulation Proof Technique*. 046. Cryptology ePrint Archive: Report 2016/046. 2016 (cited on pages 92, 93).

[Lin16b] Linux Config. *I2P - Anonymity for the Masses*. June 15, 2016. URL: https://linuxconfig.org/i2p-anonymity-for-the-masses (visited on 12/14/2016) (cited on pages 32, 151).

[Lu+05] Tianbo Lu, Binxing Fang, Yuzhong Sun, and Li Guo. "Some Remarks on Universal Re-Encryption and a Novel Practical Anonymous Tunnel". In: *Networking and Mobile Computing*. Edited by Xicheng Lu and Wei Zhao. LNCS. Zhangjiajie, China: Springer Berlin Heidelberg, August 2005, pages 853–862 (cited on page 25).

[Mat11] Nick Mathewson. *Mixminion: The Type III Remailer Protocol*. GitHub Repository. 2011. URL: https://github.com/mixminion/mixminion (visited on 09/14/2016) (cited on pages 47, 140).

[MV04] David A. McGrew and John Viega. "The Security and Performance of the Galois/Counter Mode (GCM) of Operation". In: *INDOCRYPT*. Edited by Anne Canteaut and Kapaleeswaran Viswanathan. Volume 3348. LNCS. Chennai, India: Springer, Berlin, Heidelberg, December 20, 2004, pages 343–355 (cited on page 152).

[MR10] Deepankar Medhi and Karthikeyan Ramasamy. *Network Routing: Algorithms, Protocols, and Architectures*. Elsevier. Morgan Kaufmann, July 19, 2010. 958 pages. ISBN: 978-0-08-047497-7 (cited on page 9).

[MOV96] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press. October 1996. 816 pages. ISBN: 0-8493-8523-7 (cited on pages 18, 20, 104, 151).

[Moc06] Christian Mock. *Remailer Reliability Stats [Austria]*. Web Page. October 21, 2006. URL: http://www.tahina.priv.at/~cm/stats/ (visited on 07/07/2016) (cited on page 33).

[Mol+03] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. *Mixmaster Protocol - Version 2*. IETF Internet Draft. 2003 (cited on pages 27, 28, 32, 50, 55, 63, 68).

[MN10] Tal Moran and Moni Naor. "Split-Ballot Voting: Everlasting Privacy with Distributed Trust". In: *ACM Transactions on Information and System Security* 13.2 (March 2010), 16:1–16:43 (cited on page 27).

[Mul+] Klaus G. Müller, Tony Vignaux, Ontje Lünsdorf, and Stefan Scherfke. *SimPy - Discrete Event Simulation for Python*. URL: http://simpy.readthedocs.io/en/latest/ (visited on 02/17/2017) (cited on page 137).

[MD05] S. J. Murdoch and G. Danezis. "Low-Cost Traffic Analysis of Tor". In: *Security and Privacy*. Oakland, CA, USA: IEEE Computer Society, May 2005, pages 183–195 (cited on pages 33, 46).

[Nak08] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: http://www.cryptovest.co.uk/resources/Bitcoin%20paper%20Original.pdf (visited on 03/20/2017) (cited on page 53).

[Net] NetworkX Developers. *NetworkX: High-Productivity Software for Complex Networks*. Version 1.11. GitHub Repository. URL: http://networkx.github.io/ (visited on 02/21/2016) (cited on page 138).

[Nez+09] Alireza A. Nezhad, Ali Miri, Dimitris Makrakis, and LuisO. Barbosa. "Privacy within Pervasive Communications". In: *Telecommunication Systems* 40 (3-4 2009), pages 101–116 (cited on page 83).

[nic12] nickm. *Top Changes in Tor since the 2004 Design Paper*. Blog Post. October 4, 2012. URL: https://blog.torproject.org/blog/top-changes-tor-2004-design-paper-part-1 (visited on 07/19/2012) (cited on page 43).

[NIS14] NIST. *FIPS 202, SHA-3 Standard: Permutation-Based Hash And Extendable-Output Functions*. May 28, 2014 (cited on page 20).

[OTP14] Simon Oya, Carmela Troncoso, and Fernando Pérez-González. "Do Dummies Pay Off? Limits of Dummy Traffic Protection in Anonymous Communications". In: *Privacy Enhancing Technologies*. Edited by Emiliano De Cristofaro and Steven J. Murdoch. Volume 8555. LNCS. Springer International Publishing, 2014, pages 204–223 (cited on pages 44, 45, 49).

[Pai99] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *EUROCRYPT*. Edited by Jacques Stern. Volume 1592. LNCS. Prague, Czech Republic: Springer Berlin, Heidelberg, May 1999, pages 223–238 (cited on page 21).

[Pen+06] Kun Peng, Colin Boyd, Ed Dawson, and Eiji Okamoto. "A Novel Range Test". In: *Information Security and Privacy*. Edited by Lynn Margaret Batten and Reihaneh Safavi-Naini. Volume 4058. LNCS. Springer Berlin, Heidelberg, 2006, pages 247–258 (cited on page 80).

[PTO14] F. Pérez-González, C. Troncoso, and S. Oya. "A Least Squares Approach to the Static Traffic Analysis of High-Latency Anonymous Communication Systems". In: *IEEE Transactions on Information Forensics and Security* 9.9 (September 2014), pages 1341–1355 (cited on pages 44, 45).

[PK01] Andreas Pfitzmann and Marit Köhntopp. "Anonymity, Unobservability, and Pseudonymity — A Proposal for Terminology". In: *Designing Privacy Enhancing Technologies*. Edited by Hannes Federrath. Volume 2009. LNCS. (version 0.34,2010). Springer Berlin, Heidelberg, 2001, pages 1–9 (cited on pages 4, 12, 13, 51, 52).

[Rap06] Doerte K. Rappe. "Homomorphic Cryptosystems and Their Applications". Dortmund, Germany: University of Dortmund, August 2006 (cited on page 21).

[RR98] Michael K. Reiter and Aviel D. Rubin. "Crowds: Anonymity for Web Transactions". In: *Information and System Security* 1.1 (November 1998), pages 66–92 (cited on pages 38, 79).

[RAD78] R.L. Rivest, L. Adleman, and M.L. Dertouzos. "On Data Banks and Privacy Homomorphisms". In: *Foundations on Secure Computation*. 1978, pages 169–179 (cited on page 21).

[Rog15] Phillip Rogaway. *The Moral Character of Cryptographic Work*. Cryptology ePrint Archive: Report 2015/1162. 2015 (cited on page 2).

[RS04] Phillip Rogaway and Thomas Shrimpton. "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance". In: *Fast Software Encryption*. Edited by Bimal Roy and Willi Meier. Delhi, India: Springer Berlin, Heidelberg, February 2004, pages 371–388 (cited on pages 20, 104).

[SDS02] Andrei Serjantov, Roger Dingledine, and Paul Syverson. "From a Trickle to a Flood: Active Attacks on Several Mix Types". In: *Workshop on Information Hiding*. Edited by Fabien A. P. Petitcolas. Volume 2578. LNCS. Springer Berlin Heidelberg, October 7, 2002, pages 36–52 (cited on pages 33, 68, 143, 151).

[SM05] Andrei Serjantov and Steven J. Murdoch. "Message Splitting Against the Partial Adversary". In: *Privacy Enhancing Technologies*. Edited by Danezis George and Martin David. Volume 3856. LNCS. Cavtat, Croatia: Springer Berlin Heidelberg, May 2005, pages 26–39 (cited on pages 48, 87).

[Sha79] Adi Shamir. "How to Share a Secret". In: *Communication of the ACM* 22.11 (November 1979), pages 612–613 (cited on page 83).

[SSH08] Erik Shimshock, Matt Staats, and Nick Hopper. "Breaking and Provably Fixing Minx". In: *Privacy Enhancing Technologies*. Edited by Nikita Borisov and Ian Goldberg. Volume 5234. LNCS. Leuven, Belgium: Springer Berlin Heidelberg, July 2008, pages 99–114 (cited on pages 28, 32).

[SW06] Vitaly Shmatikov and Ming-Hsiu Wang. "Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses". In: *ESORICS*. Edited by Dieter Gollmann, Jan Meier, and Andrei Sabelfeld. Volume 4189. LNCS. Springer Berlin Heidelberg, September 18, 2006, pages 18–33 (cited on pages 46, 48, 49, 96, 151).

[Sho09] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2009. 599 pages. ISBN: 978-0-521-51644-0 (cited on pages 18, 156).

[Sle13] Matt Sledge. *Bradley Manning Uncovered U.S. Torture, Abuse, Soldiers Laughing As They Killed Innocent Civilians*. August 21, 2013. URL: http://www.huffingtonpost.com/2013/08/21/bradley-manning-leaks_n_3788126.html (visited on 06/06/2017) (cited on page 2).

[Sol07] Daniel J. Solove. "'I've Got Nothing to Hide' and Other Misunderstandings of Privacy". In: *San Diego Law Review* 44 (2007), page 745 (cited on page 2).

[Ste+00] Stephen P. Smith, J. Allen Crider, Henry H. Perritt, Mengfen Shyong, Harold Krent, Larry L. Reynolds, and Stephen Menci. *Independent Technical Review of the Carnivore System*. Technical Report. IIT Research Institute, December 2000, page 117 (cited on page 12).

[Syv09] Paul Syverson. "Why I'm Not an Entropist". In: *Security Protocols XVII*. Edited by Bruce Christianson, James A. Malcolm, Vashek Matyáš, and Michael Roe. LNCS 7028. Springer Berlin Heidelberg, April 1, 2009, pages 213–230 (cited on pages 12, 145).

[THV04] Gergely Tóth, Zoltán Hornák, and Ferenc Vajda. "Measuring Anonymity Revisited". In: *Nordic Workshop on Secure IT Systems*. Edited by Sanna Liimatainen and Teemupekka Virtanen. Espoo, Finland, November 2004, pages 85–90 (cited on page 145).

[TY98] Yiannis Tsiounis and Moti Yung. "On the Security of ElGamal Based Encryption". In: *Public Key Cryptography*. Edited by Hideki Imai and Yuliang Zheng. Volume 1431. LNCS. Pacifico Yokohama, Japan: Springer Berlin, Heidelberg, February 1998, pages 117–134 (cited on page 102).

[Uni16] United Kingdom Parliament. *Investigatory Powers Act*. 2016. URL: http://services.parliament.uk/bills/2015-16/investigatorypowers.html (cited on page 1).

[Wal01] Greg Walton. *China's Golden Shield: Corporations and the Development of Surveillance Technology in the People's Republic of China*. Rights & Democracy (International Centre for Human Rights and Democratic Development), 2001. 39 pages. ISBN: 978-2-922084-42-9 (cited on page 12).

[Wei+12] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. "StegoTorus: A Camouflage Proxy for the Tor Anonymity System". In: *Computer and Communications Security*. Raleigh, North Carolina, USA: ACM Press, 2012, pages 109–120 (cited on page 152).

[WLW09] Elias Weingartner, Hendrik vom Lehn, and Klaus Wehrle. "A Performance Comparison of Recent Network Simulators". In: *International Conference on Communications*. Dresden, Germany: IEEE Computer Society, June 2009, pages 1–5 (cited on page 137).

[Wik17] WikiLeaks. *Vault 7: CIA Hacking Tools Revealed*. March 7, 2017. URL: https://wikileaks.org/ciav7p1 (visited on 03/20/2017) (cited on page 154).

[Zha+06] Yanchao Zhang, Wei Liu, Wenjing Lou, and Yuguang Fang. "MASK: Anonymous on-Demand Routing in Mobile Ad Hoc Networks". In: *IEEE Transactions on Wireless Communications* 5.9 (September 2006), pages 2376–2385 (cited on page 10).

[Zhu+04] Ye Zhu, Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. "On Flow Correlation Attacks and Countermeasures in Mix Networks". In: *Privacy Enhancing Technologies*. Edited by David Martin and Andrei Serjantov. Volume 3424. LNCS. Toronto, Canada: Springer Berlin Heidelberg, May 2004, pages 207–225 (cited on pages 46, 47).

Index

Acronyms

DDH Decisional Diffie-Hellman problem 17, 18, 22, 99, 101
DHKA Diffie-Hellman Key Agreement 20, 21, 155

HE Homomorphic Encryption 21

IK-CPA Indistinguishability of Keys under Chosen Plaintext Attacks 22, 103
IND-CCA Indistinguishability under Chosen Ciphertext Attacks 20, 21
IND-CPA Indistinguishability under Chosen Plaintext Attacks 20, 21, 102, 215
ITI Interactive Turing Machines Instances 93
IV Initialisation Vector 20, 57

KDF Key Derivation Function 20, 57

LTI Long Term Intersection Attack 44

MAC Message Authentication Code 22
MU Message Unlinkability 13
MU-session First Type of Message Unlinkability 13
MU-tracing Second Type of Message Unlinkability 13

PKE Public Key Encryption 19
PPT Probabilistic Polynomial Time 12, 18, 95, 98, 99, 102, 103, 105
PRF Pseudo-Random Function 20

RA Receiver Anonymity 13

SA Sender Anonymity 13
SKE Secret Key Encryption 19
SMPC Secure Multi-Party Computation 56, 74, 80
SU Session Unlinkability 13

TAR Traffic Analysis Resistance 13

URE Universal Re-Encryption 24, 155
USS Universal Semantic Security under Re-Encryption 24, 62, 103

Abstract

With the development of online communications in the past decades, new privacy concerns have emerged. A lot of research effort have been focusing on concealing *relationships* in Internet communications. However, most works do not prevent particular network actors from learning the original sender *or* the intended receiver of a communication. While this level of privacy is satisfactory for the common citizen, it is insufficient in contexts where individuals can be convicted for the mere sending of documents to a third party. This is the case for so-called *whistle-blowers*, who take personal risks to alert the public of anti-democratic or illegal actions performed by large organisations.

In this thesis, we consider a stronger notion of anonymity for peer-to-peer communications on the Internet, and aim at concealing the very fact that users take part in communications. To this end, we deviate from the traditional client-server architecture endorsed by most existing anonymous networks, in favor of a *homogeneous*, fully distributed architecture in which every user also acts as a relay server, allowing it to conceal its own traffic in the traffic it relays for others. In this setting, we design an Internet overlay inspired from previous works, that also proposes new privacy-enhancing mechanisms, such as the use of relationship pseudonyms for managing identities. We formally prove with state-of-the-art cryptographic proof frameworks that this protocol achieves our privacy goals. Furthermore, a practical study of the protocol shows that it introduces *high latency* in the delivery of messages, but ensures a high anonymity level even for networks of small size.

Résumé

L'avènement de l'ère digitale a changé la façon dont les individus communiquent à travers le monde, et a amené de nouvelles problématiques en terme de vie privée. La notion d'anonymat la plus répandue pour les communications sur Internet consiste à empêcher tout acteur du réseau de connaître *à la fois* l'expéditeur d'un message et son destinataire. Bien que ce niveau de protection soit adéquat pour l'utilisateur d'Internet moyen, il est insuffisant lorsqu'un individu peut être condamné pour le simple envoi de documents à une tierce partie. C'est le cas en particulier des *lanceurs d'alerte*, prenant des risques personnels pour informer le public de pratiques illégales ou antidémocratiques menées par de grandes organisations.

Dans cette thèse, nous envisageons un niveau d'anonymat plus fort, où l'objectif est de dissimuler le fait même qu'un utilisateur envoie ou reçoive des données. Pour cela, nous délaissons l'architecture client-serveur couramment utilisée dans les réseaux anonymes, en faveur d'une architecture entièrement distribuée et *homogène*, où chaque utilisateur remplit également le rôle de serveur relais, lui permettant de dissimuler son propre trafic dans celui qu'il relaie pour les autres. Dans cette optique, nous proposons un nouveau protocole pour les communications de pair à pair sur Internet. À l'aide de récents outils de preuves cryptographiques, nous prouvons que ce protocole réalise les propriétés d'anonymat désirées. De plus, nous montrons par une étude pratique que, bien que le protocole induise une grande latence dans les communications, il assure un fort anonymat, même pour des réseaux de petite taille.